

UNIVERSIDAD CARLOS III
Escuela Politécnica Superior



Ingeniería Informática

PROYECTO FIN DE CARRERA

Diseño, desarrollo y manual de una interfaz gráfica
distribuida para Maggie

Autora: Laura Romero Bachiller

Tutor: Miguel Ángel Salichs Sánchez-Caballero

Octubre de 2010

Agradecimientos

Gracias, al personal del laboratorio de robótica que me ayudó en distintos aspectos de las pruebas, o del proceso del proyecto como Ana Corrales y Fernando Alonso. Con especial mención a: Javier Gorostiza, que me facilitó los trabajos realizados hasta el momento en Flash; David García, que me explicó el funcionamiento de Maggie al más bajo nivel, y siempre disponible para ayudarme con los problemas que hubiera de conexión, o con el tabletPC; y Álvaro Castro, que se reunió conmigo en múltiples ocasiones, para explicarme las dudas que me surgían sobre Maggie y su arquitectura, y me aconsejó sobre el diseño, la estructura, y la dirección en la que llevar mi proyecto.

Gracias a Juanma por cubrirme en la beca los días que subía al laboratorio a hacer pruebas, y a Christian y sobre todo Lissar, que me ayudaron con la parte de comunicación, en los momentos de desesperación por no saber cómo podía ser posible, que en unos PCs funcionara la conexión y en otros no, con la misma versión del programa. También a Lidia y Rober, que me facilitaron sus memorias, para poder hacerme una idea de cómo esquematizar este documento, en los primeros días de su desarrollo. Y gracias en general, al resto de mis amigos que, sin entender qué estaba haciendo exactamente en muchos casos, siempre me apoyaron aunque les tuviera un poco abandonados por el trabajo, como Antonio, David, Alicia, Elena, Laura y Virginia. Y a los que no pararon de insistir en la pregunta de cuándo lo iba a terminar como Gon, Iván y Carlos.

Gracias a mi familia. A mi hermana Virginia, que me ayudó con el diseño gráfico y el juego Sapiéntino. A mi hermana Maricarmen, que a pesar de ser de letras, fue capaz de leerse todo el documento, para corregirme el formato y la escritura. Yo no fui capaz de hacer lo mismo con su doctorado... Y a mis padres, que aunque a su manera, me apoyaron para continuar siempre hacia delante a lo largo de la carrera, y a veces no se lo he sabido agradecer de la forma en que debería.

Gracias a mi tutor, Miguel Ángel. En sus clases de robótica vi claro en qué departamento quería hacer realmente el proyecto de fin de carrera, dejando un proyecto anterior, y animándome a preguntarle a él. Gracias por la increíble oportunidad de trabajar con Maggie, y por el apoyo, la confianza y también, la libertad que me dio a lo largo de la realización de este trabajo.

Y por último, gracias a mi pareja, Óscar, que siempre estuvo ahí, escuchándome cuando lo necesitaba, y ofreciéndome su ayuda y consejo. Por no dejarme flaquear, siempre hacia delante, en los momentos más difíciles, a pesar de mis enfados. No sólo en este proyecto, sino a lo largo de toda la carrera, y en la vida, durante estos años que llevamos juntos. Muchas gracias.

ÍNDICE

1	Introducción.....	1
1.1	Motivación del Proyecto.....	1
1.2	Objetivos.....	3
1.3	Estructura del Documento	3
1.4	Acrónimos.....	5
1.5	Definiciones.....	6
2	Estado de la Cuestión	8
2.1	Historia de la Robótica.....	8
2.1.1	Robots Sociales.....	14
2.1.2	Maggie.....	18
2.2	TabletsPC.....	20
2.3	Flash	22
2.3.1	Comparativa entre Versiones.....	23
2.3.2	ActionScript 3.0.....	24
3	Análisis.....	25
3.1	Requisitos de la Aplicación	25
3.1.1	Requisitos Funcionales	25
3.1.2	Requisitos No Funcionales	30
4	Diseño de la Interfaz de Usuario.....	35
4.1	Inicio y Elementos Comunes	35
4.2	Menú Principal	38
4.3	Menú de Eventos.....	39
4.4	Display de Estados.....	41
4.5	Menú de Juegos	42
4.5.1	Juego Bubbles.....	43
4.5.2	Juego Sapientino	45
4.6	Galería de Imágenes.....	49

4.7	Reproductor de Música.....	50
4.8	Reproductor de Video	52
5	Arquitectura del Programa	53
5.1	Arquitectura de Maggie	53
5.2	Comunicación TabletPC-VisMaggie.....	57
5.2.1	Seguridad.....	64
5.3	Arquitectura del Programa Flash	67
6	Manuales.....	69
6.1	Breve Manual Introductorio a Flash	69
6.1.1	El Espacio de Trabajo y las Herramientas.....	69
6.1.2	El Escenario	72
6.1.3	La línea de Tiempo y las Interpolaciones	73
6.1.4	La Biblioteca y sus Símbolos.....	78
6.1.5	Crear Botones.....	81
6.1.6	Dibujar Figuras.....	83
6.1.7	Colores y Degradados	88
6.1.8	Como Asociar Ficheros de ActionScript al Proyecto Flash.....	91
6.1.9	Creación de Proyectos y su Publicación.....	93
6.2	Breve Manual Introductorio a ActionScript 3.0.....	95
6.2.1	Primeros Pasos.....	95
6.2.2	La Orientación a Objetos	99
6.2.3	Eventos	104
6.2.4	Las Clases Movieclip y Sprite.....	107
6.2.5	Crear Formas Gráficas desde Código. Uso de la Clase Graphics.....	108
6.2.6	Crear Animaciones Mediante Código. La Clase Tween	111
6.2.7	Carga de Archivos Externos	112
6.2.8	Aplicaciones Distribuidas. Uso de Sockets.....	115
6.3	Manual de la Aplicación.....	118
6.3.1	Editar la Estructura General	118

6.3.2	Cómo Enviar/Recibir Información de Maggie.....	122
6.3.3	Cómo Usar/Modificar el Menú de Eventos.....	125
6.3.4	Cómo Usar/Modificar el Display de Estados.....	132
6.3.5	Cómo Usar/Modificar la Galería de Imágenes.....	134
6.3.6	Cómo Usar/Modificar la Lista de Reproducción de Música.....	139
6.3.7	Cómo Usar/Modificar la Reproducción de Videos.....	148
6.3.8	Cómo Hacer otra Versión del Sapientino.....	151
7	Planificación.....	159
8	Trabajos Futuros.....	161
9	Conclusiones.....	163
10	Referencias.....	166
11	Bibliografía.....	167
	Anexo: Implantación.....	169
	Pasos previos.....	169
	Ejecución.....	171

ÍNDICE DE FIGURAS

Figura 1. Pato de Jacques de Vaucanson (S. XVIII). Podía aletear, graznar e incluso “digerir” comida.....	8
Figura 2: Telar de Jacquard, un modelo de las tarjetas perforadas, y diferencial de Babbage.	11
Figura 3. Wabot-1.	12
Figura 4. Cronología de prototipos desarrollados por Honda (cortesía Honda).....	14
Figura 5. Ejemplo de robot-mascota (AIBO de Sony, 1999), y robot de limpieza.....	15
Figura 6. Kismet, cortesía del MIT	16
Figura 7. Uno de los proyectos de MDS, cortesía del MIT	17
Figura 8. Maggie	18
Figura 9. iPad cortesía de Apple.	20
Figura 10. Uso en PC con acceso a internet (%). Datos de Millward Brown (Jun. 2010).....	22
Figura 11. Diseño ventana de inicio.	35
Figura 12. Diseño ventana de aviso de error de conexión.....	36
Figura 13. Diseño inicio presentación.....	36
Figura 14. Interfaz del menú principal.	38
Figura 15. Interfaz del menú de eventos. Modelo de botón unido.....	39
Figura 16. Interfaz del menú de eventos. Modelo de botón partido.....	40
Figura 17. Interfaz del display de estados.....	41
Figura 18. Interfaz del menú de juegos.....	42
Figura 19. Interfaz de inicio del juego <i>Bubbles</i>	43
Figura 20. Interfaz dentro y al final del juego <i>Bubbles</i>	44
Figura 21. Interfaz de inicio y vista general del juego <i>Sapientino</i>	45
Figura 22. Interfaz al acertar una asociación del juego <i>Sapientino</i>	46
Figura 23. Ejemplo de selección de un elemento en el <i>Sapientino</i>	47
Figura 24. Interfaz de resultados de <i>Sapientino</i>	48
Figura 25. Interfaz de la galería de imágenes.....	49
Figura 26. Interfaz del reproductor de música.....	50
Figura 27. Interfaz del reproductor de video.....	52
Figura 28. Arquitectura Automática-Deliberativa (AD)	54
Figura 29. Arquitectura cliente-servidor.	58
Figura 30. Diagrama de flujo de datos de la habilidad.	63
Figura 31. Diagrama de clases de la aplicación Flash.	68
Figura 32. Espacio de trabajo con perfil “Clásico” de Flash CS4.....	71

Figura 33. Partes de la línea de tiempo.....	73
Figura 34. Panel de biblioteca.....	78
Figura 35. Fotogramas para animar un botón.	81
Figura 36. Modo de dibujo combinado.	83
Figura 37. Modo de dibujo de objetos.	84
Figura 38. Icono de la herramienta pluma, con la que se hacen los trazados.....	84
Figura 39. Componentes de un trazado.....	85
Figura 40. Ejemplo de dibujo con pluma. Punto curvo (izq.), y punto de vértice (der.).....	85
Figura 41. Ejemplos de cambio de inclinación.....	85
Figura 42. Personaje Akio, cortesía de www.nanoda.com	86
Figura 43. Ejemplo de remodelado de objetos.	87
Figura 44. Panel de color con degradado seleccionado.	88
Figura 45. Selección de herramienta de Transformación de degradado.....	89
Figura 46. Degradado radial (Izq.) y lineal (Der).....	90
Figura 47. Cómo asociar un archivo *.as a un símbolo Flash.....	91
Figura 48. Ventana de Configuración de Publicación.	94
Figura 49. Jerarquía de la lista de visualización.....	96
Figura 50. Ejes de coordenadas en Flash.	110
Figura 51. Ejemplo de fichero XML.	114
Figura 52. Línea de tiempo del proyecto.....	118
Figura 53. Ejemplo de botón del menú de eventos.....	126
Figura 54. Estructura contenedores de la galería de imágenes.....	135
Figura 55. Estructura del componente de volumen.	146
Figura 56. Inspector de componentes y vista previa de FLVPlayback.....	149
Figura 57. Ejemplo de recorte utilizado en el juego <i>Sapientino</i>	152
Figura 58. Ejemplo de recorte con filtro de luz, utilizado en el juego <i>Sapientino</i>	152
Figura 59. Diagrama de Gantt.....	160
Figura 60. Cliente TightVNC Viewer.	170

ÍNDICE DE TABLAS

Tabla 1. Acrónimos.....	5
Tabla 2. Definiciones	7
Tabla 3. Comparativa entre versiones de Flash. Cortesía de Adobe.....	23
Tabla 4. RF-01. Comunicación interfaz-AD	25
Tabla 5. RF-02. Navegabilidad.....	26
Tabla 6. RF-03. Activar/desactivar habilidades.....	26
Tabla 7. RF-04. Menú de Eventos Dinámico	26
Tabla 8. RF-05. Acceso al menú de eventos	27
Tabla 9. RF-06. Acceso al menú de juegos	27
Tabla 10. RF-07. Acceso al display de estados.....	27
Tabla 11. RF-08. Acceso al reproductor de música.....	27
Tabla 12. RF-09. Acceso al reproductor de video.....	27
Tabla 13. RF-10. Acceso a la galería de imágenes	28
Tabla 14. RF-11. Reutilización del juego Bubbles	28
Tabla 15. RF-12. Uso de archivos XML.....	28
Tabla 16. RF-13. Incluir juego.....	28
Tabla 17. RF-14. Display de estados	29
Tabla 18. RNF-01. Apariencia de la interfaz	30
Tabla 19. RNF-02. Idiomas.....	30
Tabla 20. RNF-03. Sistema operativo Vismaggie.....	30
Tabla 21. RNF-04. Sistema operativo tabletPC.....	30
Tabla 22. RNF-05. Uso de Flash.	31
Tabla 23. RNF-06. Respetar el formato de la AD de Maggie.....	31
Tabla 24. RNF-07. Imágenes presentación Maggie.....	31
Tabla 25. RNF-08. Tamaño de los mensajes	31
Tabla 26. RNF-09. Tamaño de letra de la interfaz.....	32
Tabla 27. RNF-10. Tamaño de los botones de la interfaz	32
Tabla 28. RNF-11. Área de la interfaz.....	32
Tabla 29. RNF-12. Intervalo de valores del display	32
Tabla 30. RNF-13. Protocolo de mensajes	33
Tabla 31. RNF-14. Repositorio de versiones	33
Tabla 32. RNF-15. Acceso tabletPC.....	34

1 Introducción

En este capítulo se explican los distintos apartados en los que se ha estructurado este documento, además de detallar la motivación que llevó a realizar este proyecto y los objetivos que se pretendían alcanzar. También se ha incluido una lista con las definiciones y acrónimos que se han considerado necesarios para comprender adecuadamente el texto.

1.1 Motivación del Proyecto

El presente proyecto tiene por objeto principal explicar cómo diseñar y desarrollar una aplicación en Flash versión CS4¹, para el Tablet PC utilizado en el robot “Maggie”, y cómo unir ésta aplicación mediante comunicación realizada con sockets, con la arquitectura que controla el robot, implementada en C++, y contenida en otro ordenador al que en adelante llamaremos “Vismaggie”. Este proyecto se ha realizado para el grupo investigador Robotics Lab², perteneciente al Departamento de Ingeniería de Sistemas y Automática, de la Universidad Carlos III de Madrid³.

Desde principios de los años 80, el Robotics Lab tiene como objetivo la investigación y el desarrollo de proyectos de alto nivel científico-tecnológico de robótica y automatización, en colaboración con empresas e instituciones tanto nacionales como internacionales.

En la actualidad hay varios proyectos en proceso de investigación, uno de los cuales está enfocado al estudio de robots sociales, con aplicaciones enfocadas tanto al entretenimiento, la educación, como a la asistencia de ancianos o personas con discapacidad psíquica. Estos robots se diseñan y ensamblan en los laboratorios del departamento, teniendo en cuenta estos condicionamientos, debido a la importancia de la interacción humano-máquina que tendrán.

“Maggie” es el principal proyecto de este tipo, y se viene desarrollando desde hace varios años enfocado principalmente al trato con niños. Este robot ha sido diseñado utilizando una arquitectura distribuida y dividida en componentes, realizada por completo en la Universidad Carlos III, según las distintas habilidades incluidas en el robot. Uno de los elementos de “Maggie” es un tabletPC incorporado en su pecho utilizado para añadir un componente visual y un mejor manejo, que acentúe la capacidad de interacción del robot.

El tablet inicial, necesitaba un puntero electrónico con el que manejarlo y no reconocía el contacto realizado con los dedos. Tampoco disponía de un software que enlazase la información recogida por este PC, con el resto de la arquitectura del robot. Sólo había una primera

¹ Adobe - <http://www.adobe.com/>

² Robotics Lab de la Universidad Carlos III de Madrid - <http://roboticslab.uc3m.es>

³ Universidad Carlos III de Madrid - <http://www.uc3m.es>

aproximación utilizando un servidor realizado en C++, añadido a un juego desarrollado en Flash versión 8, usando ActionScript 2.0, un lenguaje de desarrollo de Flash ya obsoleto, ante la nueva versión lanzada en 2009 de Macromedia Flash CS4 con ActionScript 3.0. Este programa enviaba información mediante sockets que funcionaban en una sola dirección (del tablet al robot, pero no en sentido contrario). Esto suponía una serie de problemas:

- El uso de Maggie está enfocado principalmente a los niños. Por ello, desde el punto de vista del dispositivo hardware, es más fácil para un niño utilizar una pantalla táctil con la mano, sin elementos intermedios. Tanto por seguridad (si el niño es muy pequeño), como por usabilidad, ya que intuitivamente intentará utilizar los dedos para manejarlo.
- Además, sin un programa desarrollado adecuadamente para el robot, se estará reduciendo el potencial de uso del mismo, dificultando su lanzamiento, ejecución y utilización por parte de los futuros usuarios. “Maggie” debe ofrecer una **interfaz visualmente atractiva e intuitiva** para su empleo con niños, con diversas actividades y juegos, que dé la posibilidad de activar/desactivar fácilmente las distintas habilidades que se encuentren operativas en cada momento, y sobre todo, que tenga una **comunicación interna con el robot y transparente para el usuario**, donde se pueda realizar un **tránsito asíncrono de datos en ambas direcciones**.
- Pero lo más importante, es que debido a que desde el inicio se espera que la interfaz que se va a diseñar sufra modificaciones y mejoras con el tiempo -puesto que va a formar parte de un proyecto a largo plazo, que ya lleva años en funcionamiento-, **es necesario actualizar el sistema** a la versión más reciente de Flash, usando ActionScript 3.0, y **procurar una buena documentación** con una clara estructuración de sus componentes.

Por todo ello, surgió este proyecto fin de carrera con la motivación de cambiar el tabletPC inicial, por uno realmente táctil, y de desarrollar una interfaz que sea plenamente operativa para su inmediata puesta en marcha. Esta aplicación, además, servirá de ejemplo para la realización de un manual detallado, para que de esta forma, los conocimientos aprendidos a la hora de realizar este proyecto, sirvan de guía y apoyo en un futuro y suavicen la curva de aprendizaje del personal investigador del laboratorio que desee continuar con esta labor.

1.2 Objetivos

Los principales objetivos que se pretenden alcanzar en la elaboración de este proyecto son los siguientes:

- Estudio de las funcionalidades básicas necesarias para la interfaz del tabletPC y para la realización de una nueva habilidad en Maggie, sobre la arquitectura existente.
- Búsqueda de soluciones para realizar la comunicación entre la aplicación que correrá en el tabletPC y Maggie.
- Diseño e implementación de una nueva aplicación y de un manual de desarrollador, utilizando las tecnologías seleccionadas, para el Robotics Lab.

1.3 Estructura del Documento

El documento se encuentra dividido en las siguientes secciones:

1. Introducción

Se describen los motivos por los cuales se decidió realizar el proyecto, así como los objetivos que se pretenden alcanzar. Se continúa comentando los principales apartados del documento y se finaliza con una tabla que contiene acrónimos y definiciones de interés.

2. Estado de la cuestión

Se realizará un repaso en la evolución de la robótica desde sus inicios, a las nuevas tendencias actuales de investigación, con un análisis especial en el caso de los robots denominados sociales. Se explicará desde un punto de vista general, a uno más específico, según la evolución que ha tenido el propio grupo de Robotics Lab en este ámbito.

También se expone el uso de las aplicaciones Flash, comentando sus características, y las diferencias entre versiones.

3. Análisis

- **Requisitos de usuario:** se detallan las necesidades requeridas por el Robotics Lab.
- **Requisitos software:** detalles a tener en cuenta según las limitaciones del software, y la necesidad de acoplar el programa a otro software ya hecho.

4. Diseño

En este punto se explica el funcionamiento de la interfaz de usuario y su diseño.

5. Arquitectura

- **Arquitectura Maggie:** Breve repaso a la arquitectura de Maggie.
- **Comunicación entre el tabletPC y Vismaggie:** Definición de la arquitectura elegida para la comunicación, y explicación detallada del programa servidor y las habilidades incluidas en Maggie.
- **Arquitectura del programa en Flash:** Repaso de la arquitectura general de la aplicación Flash, y su distribución por clases.

6. Manuales

- **Flash:** Se incluye un manual sobre las funcionalidades básicas en Flash necesarias para poder desarrollar una interfaz similar a la propuesta.
- **AC3:** En este punto se repasa los conocimientos generales sobre el lenguaje, necesarios para comenzar a programar, y comprender adecuadamente las clases implementadas en el proyecto.
- **Uso de la aplicación:** Este apartado se dedica a la explicación de las funcionalidades más importantes de la interfaz Flash, y cómo se hicieron, detallando su configuración a nivel de código.

7. Planificación

Se expondrá la planificación del proyecto mediante un diagrama de Gantt, y se comentará el tiempo real que ha llevado la realización del mismo.

8. Trabajos Futuros

Se comentarán una serie de ideas que han surgido a lo largo de este trabajo, para continuar con el mismo, y aumentar la interacción entre el tabletPC y Maggie.

9. Conclusiones

Se presentan las conclusiones generales y personales obtenidas tras la realización del proyecto.

10. Referencias y 11. Bibliografía

Se enumeran las referencias del proyecto y las fuentes bibliográficas.

Anexo

Se explica cómo ejecutar la aplicación, y los elementos necesarios para ello.

1.4 Acrónimos

A continuación se muestra una lista de acrónimos utilizados a lo largo del documento:

ACRÓNIMO	DEFINICIÓN
3D	3 dimensiones.
AD	Acrónimo para hacer referencia a la arquitectura Automático-Deliberativa de Maggie.
ASIMO	<i>Advanced Step in Innovative Mobility</i> . Robot humanoide creado por Honda en el año 2000.
C++	Lenguaje de programación creado a partir de C.
CMYK	<i>Cyan Magenta Yellow Key</i> . Gama de colores.
CS4	<i>Creative Suite 4</i> . Versión de Macromedia Adobe Flash
IBM	<i>International Business Machines</i> . Multinacional americana de servicios relacionados con las tecnologías de la información.
ID	<i>Identificador</i> .
FTP	<i>File Transfer Protocol</i> .
MCP	<i>Memoria a Corto Plazo</i> .
MDS	<i>Mobile/Dexterous/Social</i> . Línea de robots sociales creados en el MIT.
MIT	<i>Massachusetts Institute of Technology</i> ⁴ .
MSB	<i>Matiz Saturación Brillo</i> . Modo de color.
OO	<i>Orientado a Objetos</i> . Se utiliza en referencia a los lenguajes de programación.
P3	<i>Prototype Model 3</i> . Robot humanoide de Honda desarrollado en el año 1997.
RGB	<i>Red Green Blue</i> . Gama de colores.
RPC	<i>Remote Procedure Call</i> .
RVA	<i>Rojo Verde Azul</i> .
SRI	<i>Stanford Research Institute</i>
STRIPS	<i>Stanford Research Institute Problem Solver</i>
T3	<i>The Tomorrow Tool</i> . Primer robot comercializado, manejado por un mini-ordenador.
URL	<i>Uniform Resource Locator</i> ,
XML	<i>Extensible Markup Language</i> . Formato de almacenamiento de información de forma estructurada en forma de árbol.

Tabla 1. Acrónimos

⁴ MIT - <http://www.mit.edu/>

1.5 Definiciones

Las definiciones que se han considerado necesarias de incluir una explicación más detallada, para facilitar una mejor comprensión del documento, aparecen en la siguiente tabla:

TÉRMINO	DEFINICIÓN
ActionScript	Lenguaje de programación específico para Flash.
Capa	Las capas son como bandas de película, apiladas unas sobre otras, que contienen diferentes imágenes que se muestran en el escenario. Las imágenes de capas superiores, aparecerán por encima de las de capas inferiores.
Escena	Parte en las que se divide una película Flash, teniendo cada escena, una línea de tiempo independiente.
Flash	Programa de edición de animaciones gráficas vectoriales creado por Adobe. Se utiliza principalmente en clientes web, y es independiente del navegador.
Fotograma	Imagen particular, dentro de una sucesión de imágenes que componen una animación.
Habilidad	Una habilidad es capacidad para razonar o llevar a cabo una acción, y el elemento básico de la arquitectura AD.
Instancia	En Flash, es una copia de un símbolo que está en el escenario.
Kismet	Robot del MIT parecido a un animal, capaz de expresar distintas emociones en el rostro, y de realizar un aprendizaje similar a un bebe.
Maggie	Robot social en el que se basa este proyecto, creado íntegramente por el laboratorio de robótica de la Universidad Carlos III de Madrid.
Memoria a corto plazo	La memoria a corto plazo es la parte de la arquitectura AD donde se guarda información relevante obtenida a través de los sensores o facilitada por habilidades.
Multi-Touch	Tecnología de pantallas táctiles en las que es posible la detección precisa de múltiples contactos independientes, con los dedos de la mano, en distintas posiciones de la pantalla.
Página	Cuando hay más elementos de los que se pueden colocar en una sola ventana de la interfaz, se incluye una navegación interna, para distribuirlos y crear el efecto de estar repartidos en distintos conjuntos. A cada uno de ellos se le denomina, página.
RoboticsLab	Grupo de investigación de robótica fundado dentro del departamento de electrónica de la Universidad Carlos III.
Sandbox	Sistema de seguridad integrado en Flash, que controla la transferencia de información que puede suponer un riesgo de seguridad o privacidad.
Sapientino	Juego infantil que se basa en la unión de conceptos, como por ejemplo, nombres de objetos, con las imágenes que los representen.
Sección	Se denomina sección a cada uno de los apartados generales en los que se ha dividido la interfaz, según su función, y a las que se puede acceder, desde el menú principal.
Símbolo	Elemento que se puede animar en Flash. Hay tres instancias de símbolo: los gráficos, los clips de película, y los botones.

TÉRMINO	DEFINICIÓN
Tablet PC	Ordenador de pantalla táctil, alojado en el pecho de Maggie, encargado de soportar la interfaz gráfica del robot.
Telemetría	Tecnología que permite la medición de magnitudes físicas de forma remota, y el envío de la información recogida hacia el operador del sistema.
Vectores	En dibujo, las imágenes formadas mediante líneas y curvas, con propiedades de color y posición, que se denominan <i>vectores</i> .
Vismaggie	Ordenador contenido en Maggie, que funciona de servidor de todos los actuadores del robot, y que incluye toda la arquitectura del robot.

Tabla 2. Definiciones

2 Estado de la Cuestión

En este capítulo se realiza un repaso de la evolución de la robótica, con especial atención a los robots sociales, y el desarrollo que ha tenido Maggie a lo largo de los años que se prolonga su investigación.

También se incluye el origen de Flash, su propósito y una comparativa de características hasta llegar a nuestros días.

2.1 Historia de la Robótica

La idea de la robótica existe desde hace mucho tiempo, mucho antes de que se la definiera como tal.

Mirando en los textos griegos, alrededor del 322 a.C., Aristóteles ya introdujo un pensamiento al respecto en su cita: “si todas las herramientas, cuando se les ordene, o incluso por propia voluntad, pueden hacer el trabajo que les corresponda [...] entonces no habría ninguna necesidad de aprendices para los maestros o de esclavos para los señores” ⁽¹⁾. Una reflexión filosófica visionaria de una utopía donde los robots podrían realizar cualquier trabajo de forma autónoma.

Desde la época griega, hasta el siglo XIX, pasando por Leonardo Da Vinci y sus planos de un “prototipo” de robot humanoide, aparecieron mentes que investigaron sobre este tema, e incluso, se llegaron a construir algunos autómatas. Principalmente dedicados al entretenimiento de la nobleza, solían tratar de imitar comportamientos de animales o humanos, con complejos sistemas hidráulicos y mecánicos (imitaciones de pájaros, instrumentos musicales, figuras móviles...). Pero no fue hasta el siglo pasado cuando se puede fijar el comienzo de la robótica moderna tal y como la conocemos hoy en día, con las primeras aplicaciones de máquinas programables a la industria.

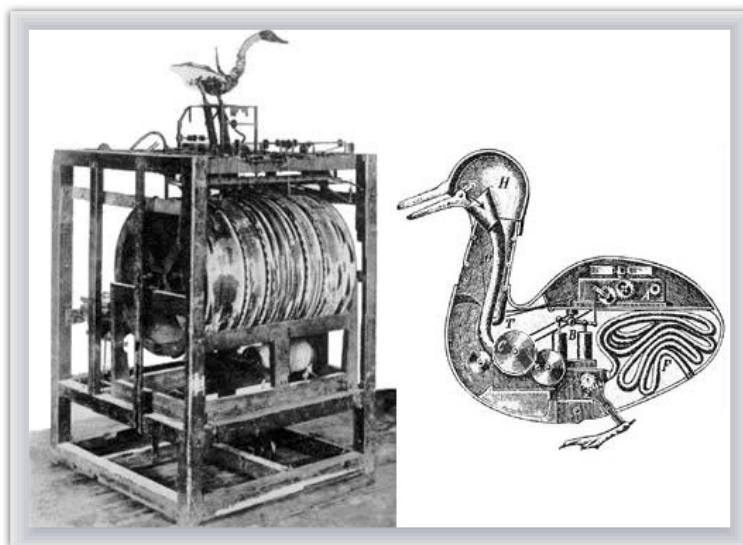


Figura 1. Pato de Jacques de Vaucanson (S. XVIII). Podía aletear, graznar e incluso “digerir” comida.

Pero, ¿qué es la robótica? Esta es una pregunta tan obvia como difícil de responder.

Si se investiga el origen de la palabra “robot”, esta aparece por primera vez en 1920 cuando el dramaturgo checo Karel Capek ⁽²⁾ uso esta palabra (derivada del checo “robota”, cuyo significado es *trabajador forzado*), en su obra *Rossum’s Universal Robots*, para referirse a un humanoide mecánico utilizado como fuerza de trabajo. Rápidamente se exportó a todos los idiomas y en la actualidad, se usa en numerosos productos (coches, robots de cocina...), como forma publicitaria de acentuar la idea de que representan un avance tecnológico en su área de negocio. Pero eso no quiere decir, que sean robots en sí.

A partir de esto, se puede plantear otra pregunta más básica que es ¿qué es un robot? Lo cierto es que no hay una definición precisa, por lo que se asume que robot es un concepto, que agrupa una serie de ideas. Un robot es:

- Un dispositivo mecánico que tiene un propósito general, y que se emplea en diversas tareas.
- Con capacidades similares o a veces superiores a los humanos.
- Que se asemeja en ocasiones al concepto de esclavos, de una forma moralmente aceptable, ya que son máquinas y no personas, al fin y al cabo.

Por lo tanto, se puede considerar, que es una evolución avanzada de las máquinas, aunque no siempre se corresponde con aparatos muy inteligentes en apariencia (por ejemplo: robots de montaje industrial).

En su origen, a las máquinas se las consideraba meras herramientas, dado que los humanos tenían que aplicar su esfuerzo para utilizarlas y controlarlas. Después, se pasó a que recibieran un aporte externo de energía, y los humanos se limitaban a ejercer el control para su correcto funcionamiento. Hoy en día, muchas están automatizadas de forma que se trata de sistemas adaptados a la aplicación y no de propósito general.

Un sistema de control automático ideal, sería un sistema universal, con capacidad de actuación y percepción para interactuar de alguna forma con el entorno. Auto transportable para que se pueda desplazar, y amigable para facilitar su uso. Además de ser escalable, y distribuido, para que pueda existir colaboración entre máquinas. Algo similar a un ser humano, aunque este tampoco sería un diseño óptimo.

Entonces, ¿qué es la robótica? Según Joseph S. Byrd: “el término de robótica se usa para describir una gran variedad de dispositivos controlados de forma remota, al igual que sistemas totalmente autónomos que denominamos robots. Está aceptado de forma general por los investigadores en el campo, que un sistema de robótica debe ser programable y debe ser capaz de llevar a cabo tareas de forma ‘autónoma’” ⁽³⁾.

Ésta es posiblemente, la definición más acertada de la robótica actual, precisamente porque no especifica nada de forma concluyente, y se adapta a las múltiples ideas de qué es exactamente la robótica. Sin embargo, de cara a las dificultades que presenta el llevar a la práctica esas ideas sobre un sistema autónomo ideal, hay mayor consenso. Los principales problemas que siempre se encuentran los investigadores a la hora de enfrentar el desarrollo de un robot del tipo que sea, son:

- **Manipulación:** lo ideal sería la capacidad de control y la percepción sensorial de unas manos, pero replicarlas es algo extremadamente complejo desde un punto de vista mecánico.
- **Transporte:** Para que sea verdaderamente autónomo, debe ser capaz de moverse por sí solo. El uso de ruedas es lo más sencillo, pero no funciona bien si el terreno es irregular. Se trata de imitar unas piernas, pero es necesario un estricto control del equilibrio y la mecánica.
- **Energía:** La autonomía también implica una forma eficiente de suministro y almacenamiento de la energía. Las baterías se gastan enseguida, por lo que para optimizar se aprovecha la energía cinética, controlando el rozamiento de las partes móviles.
- **Percepción:** Los sensores existen, pero el problema es cómo procesar la información y diferenciar objetos, asociar sonidos con ideas...
- **Interacción humano-robot:** Lo ideal sería comunicarse igual que entre dos personas. Se pueden reconocer las palabras, pero el problema radica en cómo interpretarlas correctamente. La interpretación literal del lenguaje por parte del robot, y las palabras que ofrecen información incompleta (esto, aquello, cosa...) son un grave problema, sin ir ya la entonación del mensaje (interrogativo, dubitativo, irónico...), que puede cambiar el significado de lo dicho.
- **Autonomía:** Que tenga su propia iniciativa, pero ¿dónde está el límite? ¿Qué criterio usar a la hora de elegir entre varias opciones? Ya en 1942, Isaac Asimov planteó sus famosas tres leyes de la robótica adelantándose al problema ético y moral que no se empezó a tener realmente en cuenta hasta la década de los noventa ⁽⁴⁾.
- **Inteligencia:** ¿Cómo conseguir y aplicar la inteligencia a los objetivos? ¿Qué tipo usar? (social, científica).
- **Antropomorfo:** ¿Es adecuado que tenga una forma similar a los humanos? La ventaja estaría en que podrían usar las mismas herramientas que las personas, pero llevarlo a la práctica es otra historia.

Por todo ello, el inicio de la robótica moderna, se considera fundamentalmente a partir de inicios del siglo XIX, ya que antes no eran máquinas programadas, y el objetivo y la forma de los sistemas eran diferentes. El mismo pato de Vaucanson (ver Figura 1) no tenía un comportamiento programado, sino que se ponía en marcha el sistema para que realizara un movimiento u otro.

La primera referencia a la idea moderna de robot, aparece en 1805 con el telar de Jacquard (ver Figura 2). Su inventor ideó la forma de usar tarjetas perforadas, como forma óptima de control automático del patrón del tejido, sustituyendo el trabajo que antes debía realizar una persona. Aunque parece ser que hubo precedentes, la primera comercialización de un sistema de control mediante tarjetas perforadas, fue este. Por aquel entonces, no disfrutó de una aceptación inmediata, pero se considera la inspiración de Charles Babbage en su máquina diferencial (ver Figura 2), y el inicio de la programación que más tarde desarrollaría Herman Hollerith en la empresa que más tarde se conocería como IBM⁵.

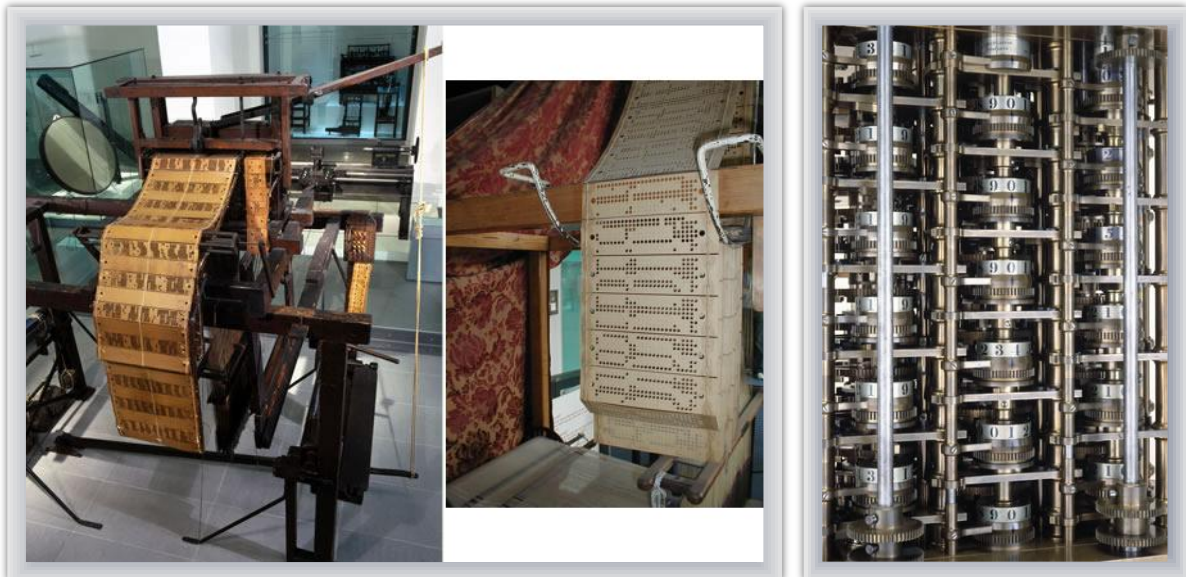


Figura 2: Telar de Jacquard, un modelo de las tarjetas perforadas, y diferencial de Babbage.

Ya en el S.XX es donde se avanza realmente con la investigación, sobre todo a partir de la década de los 50, donde el desarrollo de las técnicas de inteligencia artificial (1937, Alan Turing introduce el concepto de máquina de Turing ⁽⁵⁾ y, en 1950, su famoso test de Turing ⁽⁶⁾; 1959 se funda el Laboratorio de Inteligencia Artificial en el MIT), se empiezan a aplicar en la robótica. Se sucedieron varios hitos remarcables:

- En 1954, George Devol y Joe Engleberger diseñan el primer brazo robótico programable para realizar diversas tareas, que en 1962, se convirtió en el primer

⁵ International Business Machines - <http://www.ibm.com/es/es/>

robot industrial incluido en la cadena de montaje de automóviles de General Motors⁶. Funcionaba mediante un tambor de almacenamiento de datos magnético.

- En 1957, la Unión Soviética lanzó el Sputnik, el primer satélite artificial que se utilizó para conseguir información sobre la propagación de ondas de radio en la ionosfera, y empleó telemetría, para mediciones de temperaturas dentro y fuera del satélite.
- En 1969, ese hito fue superado por EE.UU. al enviar la primera nave tripulada a la luna, gracias a los últimos avances en ordenadores, robótica y tecnología espacial.
- En 1971 el SRI (Stanford Research Institute) mejora a su robot Shakey, de forma que pasa a estar controlado por inteligencia artificial. Dotado de una cámara y diversos sensores, utilizaba el planificador STRIPS para encontrar la ruta correcta por donde moverse a partir de la información almacenada sobre los planos de los pasillos del SRI y la información que iba obteniendo de los sensores.
- En 1973, Cincinnati Milacron Corporation sacan al mercado el T3 (“The Tomorrow Tool”), el primer robot industrial controlado por un mini-ordenador de la época (estos “mini-ordenadores” podían pesar más de 30 kilos).
- Más tarde, en el mismo año 1973 la Universidad Waseda⁷ de Japón construyó Wabot-1 (ver Figura 3), el primer robot antropomorfo a escala completa. Podía comunicarse en japonés, andar, coger objetos, y calcular distancias.

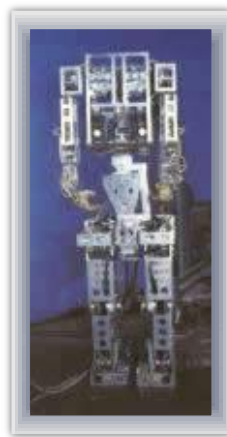


Figura 3. Wabot-1.

- En 1977, con la aparición de las películas de Star Wars, surgió toda una nueva tendencia de investigadores, que intentaban indagar sobre la posibilidad de construir un futuro donde la interacción con robots, de forma fluida y natural similar a lo mostrado en la película, fuera realidad. En esta última tendencia, es

⁶ General Motors - <http://www.gm.com/>

⁷ Universidad Waseda de Japón - <http://www.waseda.jp/top/index-e.html>

donde se inicia lo que más tarde se llamarían robots sociales. Los avances en este sentido, se comenzaron a suceder.

2.1.1 Robots Sociales.

En 1986 la empresa japonesa Honda ⁽⁷⁾, comenzó a trabajar en el desarrollo de robots humanoides con la premisa de que los robots puedan coexistir y cooperar con los humanos, realizando tareas que estos no puedan, de forma que su trabajo redunde en beneficio de la sociedad. Como se puede ver la siguiente figura, se crearon diversos prototipos, cada uno con mejor movilidad y equilibrio que el anterior, y no fue hasta 1997 con la aparición pública de P3, y más tarde de la versión mejorada de ASIMO que no se vio el verdadero avance en este campo.

Su investigación se centraba en crear una forma viable de maniobrar entre objetos de una habitación, y subir y bajar escaleras. Por ello, el diseño siempre era de robots con dos piernas, imitando a las personas, ya que si se podía crear la tecnología que controlara satisfactoriamente el equilibrio en estos casos, el robot podría ir por cualquier terreno irregular sin problemas.

En su inicio, este objetivo fue muy ambicioso, pero años más tarde, el robot ASIMO, fue el primero que podía andar y correr de forma independiente y con movimientos suaves y naturales, además de poder subir y bajar escaleras.

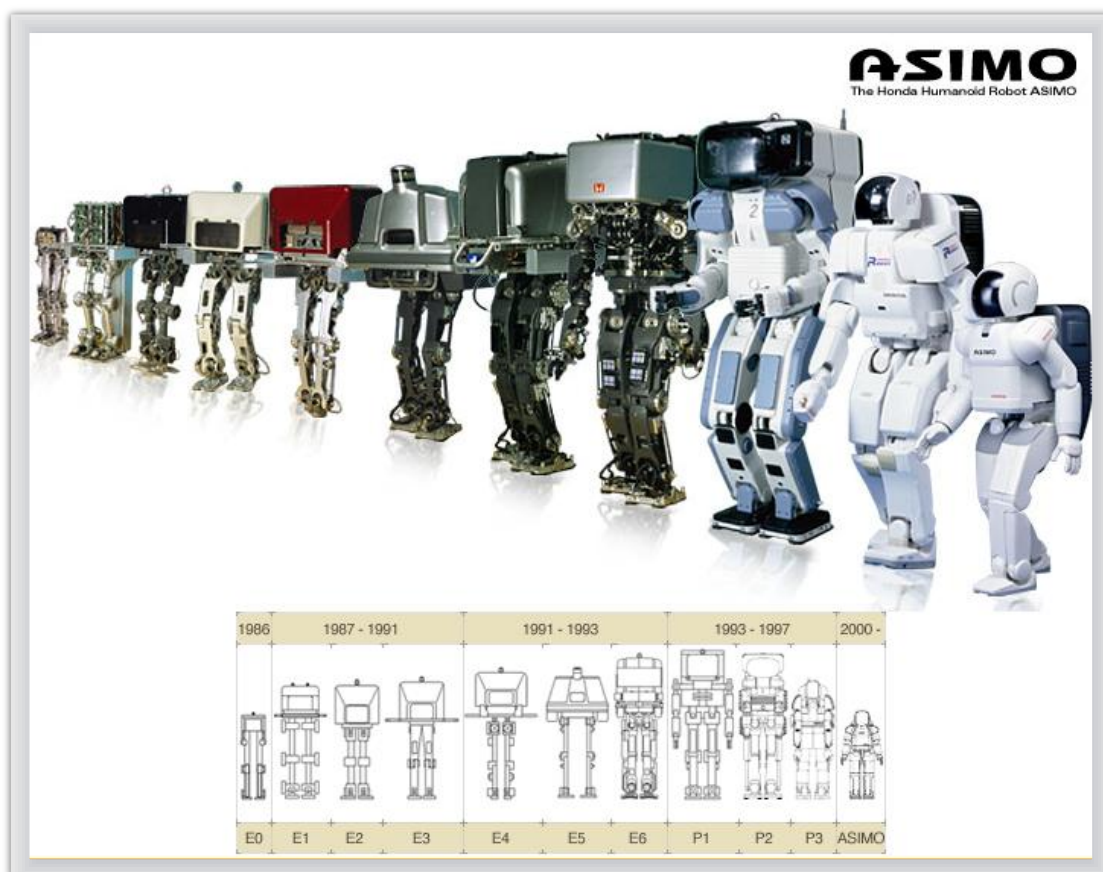


Figura 4. Cronología de prototipos desarrollados por Honda (cortesía Honda).

El robot social, surge a partir de la idea de llevar un paso más allá la robótica, de forma que se pasa de las interacciones robot-robot, a la construcción de robots que enfaticen su trato con seres humanos, y la asistencia que las máquinas pueden ofrecer. Mediante esto, los robots pasan de ser únicamente instrumentos, a ser elementos de ayuda en diferentes trabajos que pueden ser penosos o que conllevan cierto peligro o, directamente, imposibles de realizar por personas. También es posible su paso al campo del ocio y el entretenimiento. De esta forma, se comienza a pensar en robots que puedan interactuar con los humanos, de una forma sencilla y sobre todo, natural y agradable. Se piensa en robots que puedan servir de compañero de juegos de niños, o robots “enfermeros” de personas discapacitadas o que cuiden de ancianos, o que simplemente, se puedan ocupar de las tareas domésticas, para que los humanos tengan más tiempo libre que dedicar de otros quehaceres. Una comunicación fluida y una correcta comprensión, es fundamental para que no haya comportamientos extraños por parte del robot.

Pero, aunque este aspecto es el principal en este caso, los avances técnicos en este campo son de gran ayuda en robots con otros propósitos, ya que contribuyen a mejorar el control de la máquina.



Figura 5. Ejemplo de robot-mascota (AIBO de Sony⁸, 1999), y robot de limpieza.

Los robots sociales suelen tener aspectos “amigables”, que inviten a un trato directo con ellos. En caso de que se orienten a su uso con niños, se debe tener especial cuidado en esto, para que el robot no asuste a primera vista, y que sea visualmente agradable.

⁸ Web oficial de Aibo - <http://www.sonyaibo.net/home.htm>

Teniendo esto en mente, normalmente suaviza las cosas si el diseño del robot imita a un animal, o si es antropomorfo. Una vez decidido este punto, es necesario tener en cuenta otros aspectos. Hay robots que dan mayor relevancia a la movilidad y la capacidad de acción (como ASIMO), y otros en hacerlo más humanoide, trabajando en muchas ocasiones sólo con bustos, donde la investigación se centra en cómo hacer que reflejen emociones lo más humanas posibles en sus caras. Algunos ejemplos de esto último, son varias investigaciones que se están llevando a cabo en el MIT con varios robots como:

- **Kismet** (ver Figura 6): Este es un proyecto que comenzó en 1997 inspirado en la psicología de desarrollo de los niños, bajo la dirección de la Dra. Cynthia Breazell, como la principal investigadora del proyecto de máquinas sociables del MIT ⁽⁸⁾. La idea principal es que aprendiera de forma similar a un bebé, desarrollándose con la ayuda de un adulto. Es capaz de expresar sus emociones según el estado de ánimo que tiene en cada momento, el cual responde a estímulos externos. Para ello cuenta con diversos sensores visuales, auditivos y sensoriales, además de que su cabeza, ojos, cejas, labios, y orejas son móviles, para ayudar a representar el humor del robot, además de para poder enfocar mejor los sensores hacia la fuente del estímulo.

El que sea humanoide, no es algo arbitrario. Según el profesor Rodney Brooks, director del Laboratorio de Inteligencia Artificial del MIT, un robot artificial debe tener apariencia humana para facilitar la interacción con ellos de igual a igual ⁽⁹⁾.

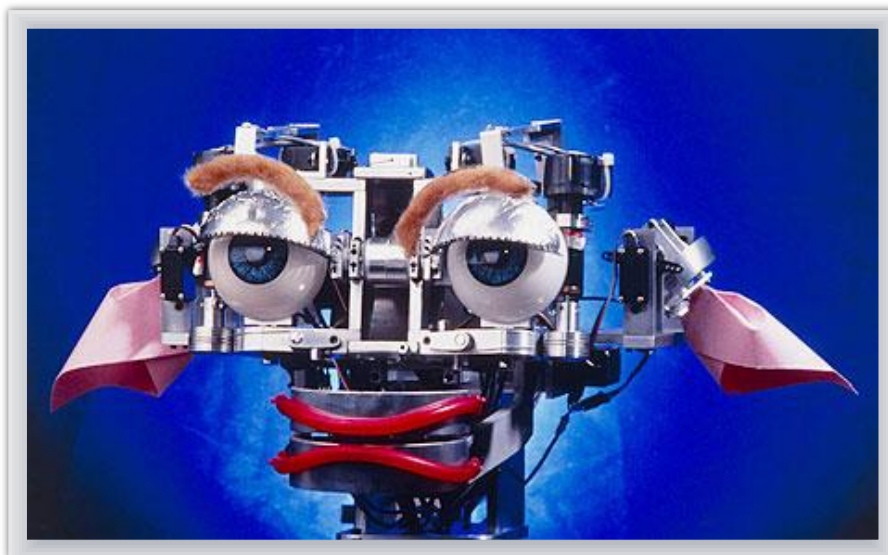


Figura 6. Kismet, cortesía del MIT

- **MDS** (Mobile/Dexterous/Social): MDS es una línea de robots creados en el grupo de Robots Personales del MIT ⁽¹⁰⁾, que disponen de movilidad, cierta destreza y habilidades de comunicación (verbal y no verbal), e interacción social con humanos. Su propósito es el de crear un sistema para robots, capaces de interactuar de forma natural, que aprendan y que cooperen con las personas. El proporcionarlos de una

expresividad que los haga más humanos, al imitar expresiones faciales para diferentes estados de ánimo, es un medio para lograr ese objetivo. El diseño general se realiza en el MIT, pero su sistema integrado es manejado por Xitome Design⁹. Dispone de cabeza, cejas, ojos, párpados, y boca móviles, además de unos brazos y una base con ruedas, para permitir su movilidad por el entorno.

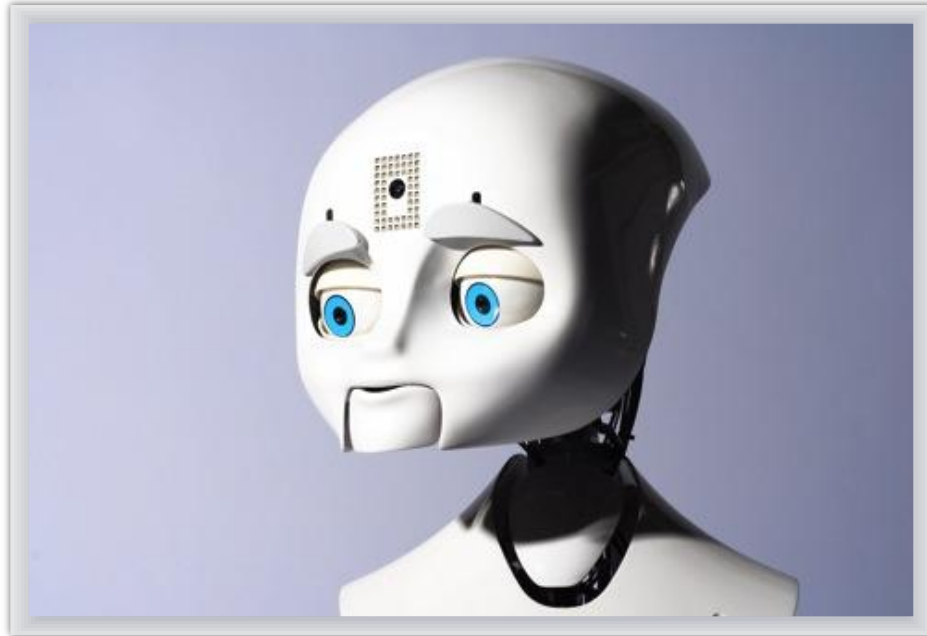


Figura 7. Uno de los proyectos de MDS, cortesía del MIT

⁹ Xitome Design - <http://www.xitome.com/>

2.1.2 Maggie

Maggie, es la plataforma robótica para la que se ha desarrollado este trabajo. Es un robot social, de 1.35 metros de altura, y apariencia amigable. Está diseñada y creada tanto a nivel de software como de hardware, en el Laboratorio de Robótica de la Universidad Carlos III de Madrid, para investigación en la interacción humano-robot (Salichs & all, 2006). Está enfocado a su uso con personas discapacitadas, como asistencia personal, o para su interacción con niños.



Figura 8. Maggie

Su arquitectura viene explicada en detalle en el punto 5.1 Arquitectura de Maggie. Pero a modo de resumen, Maggie cuenta con un ordenador táctil en el pecho y otro ordenador que hace de servidor de todo el software de control de las habilidades de Maggie, y de control de las ruedas de la base, visión, voz, sensores, telémetro láser, ultrasonidos, infrarrojos y para los diversos actuadores de la cabeza, brazos y párpados. Los ordenadores están interconectados por medio de una red Ethernet 802.3 y con una red WiFi 802.11 para conectar con el exterior.

Su investigación no se centra tanto en el movimiento como ASIMO, o en la expresividad facial, como Kismet o MDS, aunque no olvida ninguno de estos campos. Maggie es capaz de moverse con autonomía por los pasillos y rampas de la universidad, y también está dotada de un sistema de voz, capaz de hacer reflejar distintos estados de ánimo usando una entonación u otra. El movimiento de sus brazos, cabeza y párpados, y la iluminación de los leds azules integrados en su boca, que se activan de forma sincronizada a su voz, acompañan su puesta en escena para enfatizar en el aspecto de la comunicación no verbal. Los sensores de tacto repartidos por todo

su cuerpo, permiten que se realicen respuestas sensoriales por el contacto, como reírse si la hacen cosquillas.

Actualmente, se sigue investigando en dotarla de mayor expresividad gráfica y visual, a través de la comunicación con el ordenador táctil de su pecho, mediante la interfaz flash, cuya primera versión se ha realizado con este proyecto de fin de carrera. Los elementos añadidos a esa interfaz, servirán como introducción para que en el futuro, entre otras cosas, Maggie pueda mostrar de forma gráfica (por ejemplo, usando una especie de ecualizador), sus estados de ánimo y necesidades.

También se trabaja continuamente en dotarla de un mayor número de habilidades (diferentes capacidades para razonar o llevar a cabo una acción), de forma que pueda realizar un gran abanico de funciones.

2.2 TabletsPC

Un TabletPC es un ordenador portátil, que dispone de pantalla táctil donde el usuario puede trabajar sin necesidad de teclado y ratón, usando un puntero, o bien directamente los dedos, según como sea el tipo de pantalla.

Los hay que sólo tienen la pantalla táctil, como el nuevo iPad de Apple¹⁰, siendo muy ligero, o que son ordenadores portátiles normales, cuya pantalla se puede rotar y colocar tapando el teclado, para utilizar como si fuera una pizarra. Este último tipo, son los llamados portátiles convertibles, y se corresponde con el modelo que se está usando actualmente en Maggie.



Figura 9. iPad cortesía de Apple.

Por regla general, son dispositivos que utilizan procesadores que consuman poca energía, para que la batería perdure durante un período más largo de tiempo. Requieren de un sistema operativo que proporcione el software especial necesario para permitir trabajar a través de la pantalla táctil. En cuanto al hardware necesario de las pantallas, los dos tipos más usados son:

- **Resistivas:** Están formadas por múltiples capas, dos de ellas compuestas de material conductor, entre las cuales hay una pequeña separación. Mediante presión de cualquier objeto, el lugar donde unan las capas conductoras, será donde se detecte la

¹⁰ iPad de Apple - <http://www.apple.com/es/ipad/>

pulsación. Admiten uso de puntero o los dedos, son más asequibles y no se ven afectadas por el polvo o el agua. Sin embargo, la respuesta es lenta, se rayan más fácilmente y tienen hasta un 25% menos de brillo que las normales debido a las capas necesarias, además de que no sirven para detectar múltiples contactos simultáneos.

- **Capacitivas:** Estas pantallas están cubiertas de un material conductor de corriente eléctrica continua, por lo que ante el contacto con un dedo o un objeto que tenga carga eléctrica, se detecta la pulsación. La presión no influye aquí, y se pueden detectar varios contactos, además de que tienen una alta claridad. Como inconvenientes, al requerir de un complejo control de la carga eléctrica de la pantalla, tiene un precio elevado y puede traer una pérdida de precisión en aparatos pequeños al usar los dedos.

Debido principalmente al hardware de las pantallas, el coste total de un dispositivo táctil es casi el doble de lo que valdría un aparato convencional con un nivel equivalente de procesador, disco duro, memoria, etc. Por este motivo, en general las compañías no han apostado fuerte por este formato de ordenadores, cuyo precio es mayor que el de otros dispositivos, ya que se pueden comprar portátiles con mayor potencia en sus componentes hardware, y más baratos que uno táctil, y la mayoría de los consumidores no considera que compense. Pero de cara a su uso en el campo de los negocios, y en algunos casos en los que se quiera subrayar el componente interactivo y moderno de la aplicación, sí se reconoce su utilidad, como es el caso de la interfaz que se ha diseñado para este proyecto.

2.3 Flash

Adobe Flash es una plataforma de software multimedia, que se usa para añadir interactividad, animación y video a las páginas web. Es un software que requiere de licencia, bajo el copyright de Adobe Systems Inc¹¹. Se usa mucho en juegos, anuncios y animaciones web, aunque también se pueden crear aplicaciones ejecutables desde Windows.

Por un lado está:

- **Adobe Flash Player:** Es el reproductor que se usa para poder visualizar el contenido creado con Flash. Como se puede ver en la siguiente figura, es la plataforma más extendida en su uso.

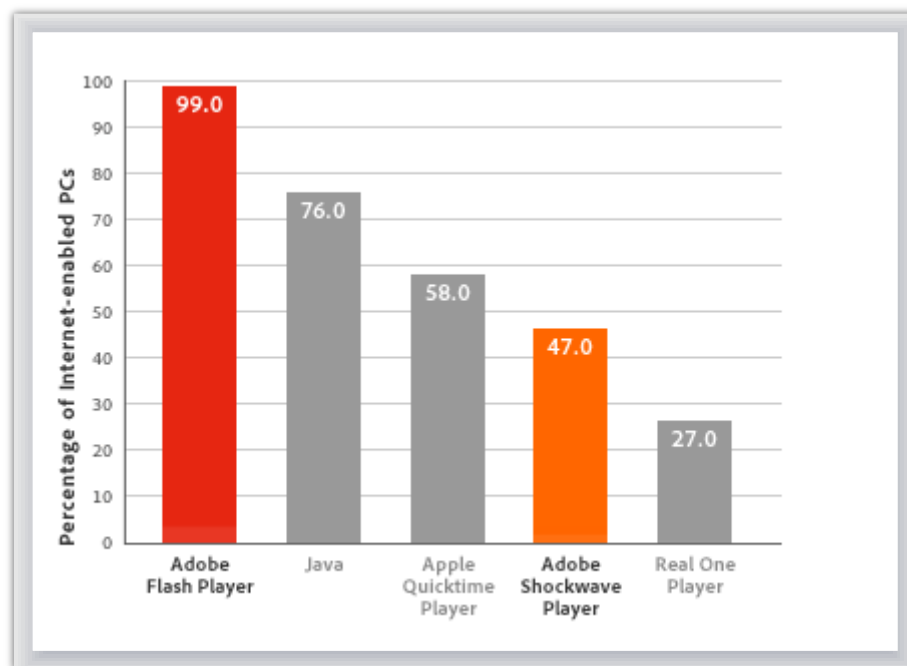


Figura 10. Uso en PC con acceso a internet (%). Datos de Millward Brown (Jun. 2010)

- **Adobe Flash:** Es la aplicación que se usa para desarrollar los contenidos Flash. Tiene su propio lenguaje de programación denominado ActionScript, que se usa para manejar los eventos, los componentes y las ejecuciones de las animaciones hechas en Flash. La última versión en salir al mercado fue Adobe Flash Professional Creative Suite 5, a mediados de 2010.

¹¹ Adobe Systems Inc. - <http://www.adobe.com>

2.3.1 Comparativa entre Versiones

A continuación, se incluye una comparativa entre las últimas versiones de la herramienta de desarrollo para Flash, que Adobe ha sacado al mercado.




			
	Flash CS5 Professional	Flash CS4 Professional	Flash CS3 Professional
Procesador de textos	✓	✗	✗
Ficheros fuente basados en XML	✓	✗	✗
Panel de Code Snippets	✓	✗	✗
Integración de Adobe Flash Builder™	✓	✗	✗
Spring for Bones	✓	✗	✗
Herramienta de dibujo Deco	✓	✗	✗
Animación basada en objeto	✓	✓	✗
Transformación 3D	✓	✓	✗
Cinemáticas con la herramienta Huesos	✓	✓	✗
Modelado procedural con Deco y Spray Brush	✓	✓	✗
Herramientas de video sofisticadas	✓	✓	✗
Adobe Creative Suite user interface	✓	✓	✗
Soporte XFL	✓	✓	✗
Animación convertida a ActionScript	✓	✓	✓
Copiado y pegado en animación	✓	✓	✓
ActionScript 3.0	✓	✓	✓
Efectos de filtrado	✓	✓	✓

Tabla 3. Comparativa entre versiones de Flash. Cortesía de Adobe.

Las versiones anteriores de Flash 8 no se incluyen, debido a su antigüedad, y a que utilizaban las versiones 1.0 y 2.0 de ActionScript, que se están dejando de usar desde la aparición de ActionScript 3.0.

2.3.2 *ActionScript 3.0*

El salto que se produjo a partir de Adobe Flash CS3, fue inmenso con la remodelación del lenguaje de programación que se venía usando. Paso de incluirse el código directamente en los fotogramas del escenario principal, e incluso, en los fotogramas de cada componente de la animación, a poder separar el código en clases con extensión *.as, adquiriendo el paradigma de programación orientada a objetos. Esto proporcionaba una mayor claridad a la hora de programar, asociando cada clase al elemento concreto al que iba a controlar, y quitando peso de la animación, ya que el código se encuentra dividido en ficheros, y no cuelga directamente de los fotogramas, lo que hace que las animaciones sean más rápidas.

Otras diferencias respecto a versiones anteriores se remarcen en el capítulo 6.2 Breve Manual Introductorio a ActionScript 3.0.

3 Análisis

En este capítulo se procede al estudio inicial de los requisitos que se tuvieron en cuenta en el desarrollo del programa, y un resumen con las decisiones de diseño que se tomaron respecto a la estructura del programa, en sucesivas reuniones con el personal investigador del laboratorio.

3.1 Requisitos de la Aplicación

A continuación se especifican los principales requisitos de la aplicación, divididos en dos categorías, funcionales y no funcionales. Los requisitos se han recogido en tablas, donde aparecen indicados los siguientes campos:

- **ID.** Identifica de forma unívoca un requisito. Debe seguir la siguiente nomenclatura: RF-XX y RNF-XX, requisito funcional y no funcional respectivamente, siendo XX números comprendidos entre 0 y 9, incrementándose en una unidad consecutivamente con cada requisito, comenzando en 01. También formará parte del identificador un nombre que resuma el requisito.
- **Descripción.** Especificación detallada del requisito.
- **Necesidad.** Relevancia del requisito para el resultado global de la aplicación. Puede tomar los valores “Esencial”, “Conveniente” u “Opcional”.
- **Estabilidad.** Especifica la susceptibilidad del requisito a ser modificado. Puede tomar los valores de “Estable” o “No estable”.

3.1.1 Requisitos Funcionales

Lista de requisitos que especifican las distintas tareas que debe desempeñar la aplicación en la práctica.

ID: RF-01	Comunicación interfaz-AD
<i>Descripción</i>	La interfaz debe de poder comunicarse con la arquitectura de Maggie, que está contenida en otro ordenador. Por lo tanto, se debe de diseñar una aplicación distribuida y el método que se emplee de comunicación, debe de permitir el poder enviar mensajes en ambas direcciones.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 4. RF-01. Comunicación interfaz-AD

ID: RF-02	Navegabilidad
<i>Descripción</i>	La interfaz debe proporcionar navegabilidad, es decir, la facilidad con la que el usuario se desplaza por las páginas que la componen.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 5. RF-02. Navegabilidad

ID: RF-03	Activar/desactivar habilidades
<i>Descripción</i>	Debe existir una sección que contenga un listado de las habilidades operativas de Maggie, y que ofrezca la posibilidad de activar o desactivar cualquiera de ellas, de forma inmediata, a través de la interfaz.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 6. RF-03. Activar/desactivar habilidades

ID: RF-04	Menú de Eventos Dinámico
<i>Descripción</i>	<p>El menú de eventos debe de generarse de forma dinámica, considerando que:</p> <ul style="list-style-type: none"> • El número de elementos será variable, lo que se deberá tener en cuenta a la hora de distribuirlos en la pantalla. • El texto, los iconos, y la información sobre el evento se leerá de un archivo aparte, que debe poder ser modificado por el usuario. • Se darán dos opciones de formato de la interfaz: <ul style="list-style-type: none"> ○ Una en la que para cada evento se disponga de un botón de activado y otro de apagado. ○ Otra donde el botón al pulsarlo, cambie de ON a OFF en cada caso, y se mantenga una memoria de aquellos que se hayan pulsado.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 7. RF-04. Menú de Eventos Dinámico

ID: RF-05	Acceso al menú de eventos
<i>Descripción</i>	El menú de eventos debe ser accesible desde la página principal.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 8. RF-05. Acceso al menú de eventos

ID: RF-06	Acceso al menú de juegos
<i>Descripción</i>	El menú de juegos debe ser accesible desde la página principal.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 9. RF-06. Acceso al menú de juegos

ID: RF-07	Acceso al display de estados
<i>Descripción</i>	El display de estados debe ser accesible desde la página principal.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 10. RF-07. Acceso al display de estados

ID: RF-08	Acceso al reproductor de música
<i>Descripción</i>	El reproductor de estados debe ser accesible desde la página principal.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 11. RF-08. Acceso al reproductor de música

ID: RF-09	Acceso al reproductor de video
<i>Descripción</i>	El reproductor de video debe ser accesible desde la página principal.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 12. RF-09. Acceso al reproductor de video

ID: RF-10	Acceso a la galería de imágenes
<i>Descripción</i>	La galería de imágenes debe ser accesible desde la página principal.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 13. RF-10. Acceso a la galería de imágenes

ID: RF-11	Reutilización del juego Bubbles
<i>Descripción</i>	El programa debe de incluir una versión actualizada al lenguaje ActionScript 3.0 y utilizando programación O.O. del juego “Bubbles”.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 14. RF-11. Reutilización del juego Bubbles

ID: RF-12	Uso de archivos XML
<i>Descripción</i>	La interfaz debe de obtener toda la información que vaya a ser configurable por el usuario, a partir de archivos XML, que serán los que hagan la función de base de datos del sistema. De esta forma, se obtendrá la información para generar el menú dinámico de eventos, la lista de reproducción de canciones, la de video, y la galería de imágenes, además de otras informaciones relevantes de los juegos que sean susceptibles de futuras modificaciones.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 15. RF-12. Uso de archivos XML

ID: RF-13	Incluir juego
<i>Descripción</i>	La interfaz debe de contener un juego realizado en Flash, que ponga de manifiesto de alguna manera la comunicación entre el tabletPC y Maggie. La temática de este juego es abierta, pero debe de tenerse en cuenta que ha de ser para niños.
<i>Necesidad</i>	Opcional
<i>Estabilidad</i>	No estable

Tabla 16. RF-13. Incluir juego

ID: RF-14	Display de estados
<i>Descripción</i>	La interfaz debe de contener un apartado que se pueda usar en un futuro para mostrar, de forma gráfica en un eje, diversos estados de Maggie. El valor dependerá de mensajes que lleguen por el socket a la aplicación Flash, desde Maggie, y se deberá actualizar de forma inmediata, cambiando la altura y color del gráfico.
<i>Necesidad</i>	Conveniente
<i>Estabilidad</i>	No estable

Tabla 17. RF-14. Display de estados

3.1.2 Requisitos No Funcionales

Lista de requisitos que tienen relación con aspectos menos prácticos de la aplicación, que no especifican que tareas debe de realizar el programa, sino puntos indirectos a tener en cuenta en el diseño.

ID: RNF-01	Apariencia de la interfaz
<i>Descripción</i>	La apariencia de la interfaz debe tener un diseño con colores llamativos y animaciones, para que sea visualmente atractiva para los niños. Tendrá que tener el logo de la Universidad y el del RoboticsLab.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 18. RNF-01. Apariencia de la interfaz

ID: RNF-02	Idiomas
<i>Descripción</i>	Debe de haber dos versiones de la interfaz, una con el texto en español, y otra en inglés para utilizar en los eventos internacionales en los que participe Maggie.
<i>Necesidad</i>	Conveniente
<i>Estabilidad</i>	Estable

Tabla 19. RNF-02. Idiomas

ID: RNF-03	Sistema operativo Vismaggie
<i>Descripción</i>	El sistema operativo de VisMaggie será una versión Fedora de Linux.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 20. RNF-03. Sistema operativo Vismaggie.

ID: RNF-04	Sistema operativo tabletPC
<i>Descripción</i>	El sistema operativo del tabletPC será un Windows Vista Home.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 21. RNF-04. Sistema operativo tabletPC.

ID: RNF-05	Uso de Flash
<i>Descripción</i>	La interfaz gráfica que se desarrolle en el tabletPC debe de implementarse utilizando la tecnología Flash.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 22. RNF-05. Uso de Flash.

ID: RNF-06	Respetar el formato de la AD de Maggie
<i>Descripción</i>	El programa que haga de enlace en Vismaggie para la comunicación con el tabletPC, deberá estar implementado en C++, y seguir el formato definido en la AD de cara a los nombres de las clases, funciones y variables.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 23. RNF-06. Respetar el formato de la AD de Maggie.

ID: RNF-07	Imágenes presentación Maggie
<i>Descripción</i>	Las imágenes utilizadas para realizar la animación del modo pausa de la interfaz, deben de ser las mismas, que las utilizadas en el archivo de PowerPoint que se venía utilizando para este fin, antes de la implantación de la interfaz Flash.
<i>Necesidad</i>	Opcional
<i>Estabilidad</i>	No estable

Tabla 24. RNF-07. Imágenes presentación Maggie.

ID: RNF-08	Tamaño de los mensajes
<i>Descripción</i>	El tamaño de los mensajes enviados del tabletPC a VisMaggie y viceversa, deberá ser de un máximo de cien caracteres.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	No estable

Tabla 25. RNF-08. Tamaño de los mensajes

ID: RNF-09	Tamaño de letra de la interfaz
Descripción	El tamaño de la letra del texto que aparezca en la interfaz, debe de ser lo más grande posible, para que se pueda ver estando a una cierta distancia del robot.
Necesidad	Conveniente
Estabilidad	Estable

Tabla 26. RNF-09. Tamaño de letra de la interfaz

ID: RNF-10	Tamaño de los botones de la interfaz
Descripción	El tamaño de los botones debe de ser el adecuado, y deben estar bien ordenados en la interfaz, para que se pueda seleccionar cómodamente el botón deseado pulsando con los dedos en la pantalla táctil.
Necesidad	Esencial
Estabilidad	Estable

Tabla 27. RNF-10. Tamaño de los botones de la interfaz

ID: RNF-11	Área de la interfaz
Descripción	El área de la ventana del programa flash, debe de ocupar exactamente el hueco de 15x21 centímetros, del pecho de Maggie. Esta área es menor que el tamaño del portátil, por lo que la posición de la interfaz, y su tamaño se debe de ajustar para que ocupe exactamente ese tamaño, en la posición adecuada. Esto es para aprovechar el máximo sitio disponible y, para que no haya elementos interactivos que se encuentren fuera de la zona accesible del tabletPC.
Necesidad	Esencial
Estabilidad	Estable

Tabla 28. RNF-11. Área de la interfaz

ID: RNF-12	Intervalo de valores del display
Descripción	Los valores que reciba el display de estados estarán dentro del intervalo [-100, 100].
Necesidad	Conveniente
Estabilidad	No estable

Tabla 29. RNF-12. Intervalo de valores del display

ID: RNF-13	Protocolo de mensajes
Descripción	<p>Los mensajes que se envíen por el socket, deben seguir un determinado protocolo para su correcta comprensión en ambas partes del programa (la contenida en Flash, y la contenida en Vismaggie). Los datos estarán separados por “:”, y el primer elemento indicará el tipo de información que contenga el mensaje. Los esquemas a seguir son:</p> <ul style="list-style-type: none"> Mensajes sobre eventos: <i>ev:id_evento:parámetro_evento</i> Mensajes sobre el display: <i>display:valor_display</i> <p>Cuando se muestren varios a la vez, se podrá utilizar: <i>display:valor_display1:valor_display2:....valor_displayN</i></p> <ul style="list-style-type: none"> Mensajes de voz, para que Maggie diga algo en español: <i>di:frase en español</i> Mensajes de voz, para que Maggie diga algo en inglés: <i>say:frase en inglés</i> Para cerrar el socket. <i>cerrar</i> Para otros datos no contenidos en esta práctica: <i>Mensaje de texto, que se copiará en MCP, dando aviso a todas las habilidades suscritas, que se deberán de encargar de ver lo que se ha puesto en la posición correspondiente de MCP.</i>
Necesidad	Conveniente
Estabilidad	No estable

Tabla 30. RNF-13. Protocolo de mensajes

ID: RNF-14	Repositorio de versiones
Descripción	El programa contenido en Vismaggie, debe de añadirse al repositorio principal de la arquitectura de Maggie, tan pronto como se tenga una versión robusta y definitiva del código C++.
Necesidad	Conveniente
Estabilidad	No estable

Tabla 31. RNF-14. Repositorio de versiones

ID: RNF-15	Acceso tabletPC
<i>Descripción</i>	Debido a que el tabletPC se encuentra instalado dentro de la carcasa del robot, y es de difícil acceso, se tiene que utilizar un acceso vía escritorio remoto o similar, para poder tener acceso a él. Entre otras cosas, al guardar las versiones del programa Flash, o cuando se realicen las pruebas en el laboratorio. Debido al tipo de sistema operativo del tabletPC (los Windows home no admiten conexiones entrantes de escritorio remoto), se ha instalado el programa TightVNC, para permitir una comunicación cliente-servidor. El servidor se lanza al inicio de Windows en el tabletPC, y permite una conexión análoga de escritorio remoto, al ejecutar desde otro ordenador el cliente del programa a partir de la ip de destino y una contraseña.
<i>Necesidad</i>	Esencial
<i>Estabilidad</i>	Estable

Tabla 32. RNF-15. Acceso tabletPC

4 Diseño de la Interfaz de Usuario

Una vez efectuado el estudio inicial de requisitos, se procede a mostrar el diseño de la interfaz de usuario. Esta interfaz realizada en Flash para el tabletPC de Maggie consiste en una serie de menús y secciones, en los que se ha dividido la aplicación para ofrecer una navegabilidad intuitiva y muy sencilla.

A continuación se pasará a identificar los elementos interactivos de cada apartado de la interfaz, detallando su funcionalidad.

4.1 Inicio y Elementos Comunes

El diseño general responde a la idea de que fuera una interfaz atractiva, sin caer en decoraciones excesivas, donde predominaran los tonos azulados, para no resaltar con el aspecto exterior de Maggie, y utilizar un tipo de letra informal, que sea atrayente para los niños.

En la Figura 11 se muestra cómo es la primera ventana que ve el usuario. Al ejecutar la aplicación, aparece esta imagen, donde se iniciará una breve animación con Maggie saludando con su brazo, quedando inmóvil después. De este modo, el programa queda a la espera de que el usuario pulse el botón con el texto *Conectar*, para iniciar la petición de conexión mediante sockets a Maggie, y pasar al menú principal de la aplicación (ver Figura 14).



Figura 11. Diseño ventana de inicio.

Si el programa servidor no está lanzado en Maggie, o hay algún problema con la comunicación, todos los procesos que dependan de enviar o recibir información a Maggie, tendrán su funcionalidad reducida. Se mostrará un aviso en este caso, para avisar al usuario:



Figura 12. Diseño ventana de aviso de error de conexión.

Un elemento interactivo importante a destacar, es el símbolo del laboratorio de robótica. Este icono aparece no sólo en la Figura 11, sino que está presente en todo momento en la interfaz, en la misma posición en la esquina inferior derecha de la ventana.

Su funcionalidad reside en que al pulsar sobre él, deja en pausa todo lo que se estuviera haciendo en ese momento en la interfaz, e inicia la visualización de una presentación sobre Maggie. Esa presentación (ver Figura 13), se ha realizado usando animaciones de objetos en Flash, en base a las imágenes facilitadas por el personal investigador del laboratorio. La presentación está en un bucle infinito, por lo que cuando se quiera cerrar y volver a la interfaz, basta con hacer un clic con el ratón en cualquier punto de la pantalla.



Figura 13. Diseño inicio presentación.

Otro elemento importante presente en todas las ventanas de la interfaz, a excepción de la ventana de inicio y el menú principal, es el **botón “atrás”** (ver Figura 15). Este botón tiene forma de una flecha grande azul, situada en la esquina inferior izquierda y sirve para regresar al menú principal desde cualquier sección. En caso de que el usuario este en un juego, volverá al menú de juegos, y desde ahí, se podrá pulsar de nuevo, para ir al menú principal.

4.2 Menú Principal

En la Figura 14, se puede observar la lista con las distintas opciones disponibles en la interfaz. Se puede acceder a cada sección, pulsando en el botón dinámico colocado a la izquierda del texto que le corresponda.

Una de las secciones no es plenamente funcional aún. Éste es el caso del display de estados, que corresponde a la parte gráfica de un programa en el que está trabajando personal investigador del laboratorio en la actualidad. Aun así, se ha aportado esta sección para facilitar el avance de la aplicación en desarrollo.

Otras secciones como: la galería de imágenes, la sección musical y el video, se han querido añadir como muestra de lo que puede ofrecer Flash. También han servido a modo de ejemplo en el manual (ver el punto 6.3 Manual de la Aplicación), para explicar cómo reproducir videos, pistas de audio, y como cargar imágenes dinámicamente, además del uso de diversos efectos de animación. Este conocimiento se ha considerado de utilidad en futuras ampliaciones del presente trabajo, o si se desean modificar las existentes, para combinarlo con habilidades de Maggie, y darle mayor interactividad a la interfaz.


Un elemento remarcable del menú principal, es el botón de cierre . Cuando el usuario pulsa este botón, se cierra la aplicación, además de la comunicación por sockets.



Figura 14. Interfaz del menú principal.

4.3 Menú de Eventos

Este es el menú diseñado como controlador gráfico de las habilidades de Maggie (ver Figura 15 y Figura 16). Se listan los nombres de cada una de ellas, con un icono y un botón asociados. De esta forma, se pueden enviar los eventos de activación y desactivación de esas habilidades mediante un simple clic de ratón.

Se han diseñado dos tipos de botones diferentes, según la opción elegida por el usuario al publicar el documento flash (ver punto 6.3.3 Cómo Usar/Modificar el Menú de Eventos):

- **Modelo Unido** (Figura 15): En este caso, solo hay un botón para cada habilidad. Por defecto se muestra al principio preparado para enviar la orden de activación. Al pulsarlo por primera vez, el botón cambiará, y aparecerá el texto *OFF*, ya que si se pulsa el mismo botón una segunda vez, se enviará el evento de desactivación de la habilidad. Con ese segundo clic, el botón cambiará de nuevo al texto *ON*, y quedará preparado como en un principio.

Debido al cambio gráfico que se produce con cada pulsación en un botón, durante la navegación interna de la sección, se recordará el último estado de cada botón. Esta memoria desaparecerá al volver al menú principal.

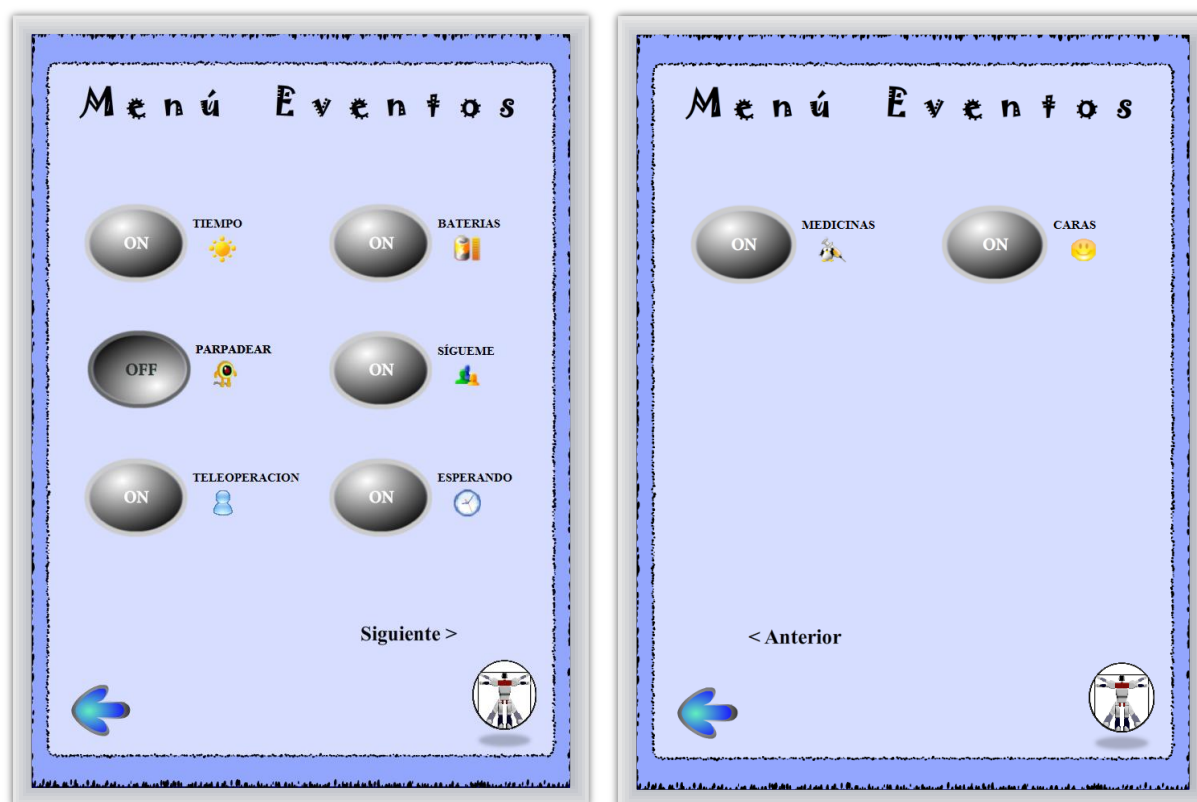


Figura 15. Interfaz del menú de eventos. Modelo de botón unido.

- **Modelo Partido** (Figura 16): Este diseño presenta dos botones para cada habilidad, de forma que siempre están presentes las opciones de encendido y apagado, evitando posibles inconvenientes en caso de cambios de sección. Al pulsar con el ratón, se enviará la orden adecuada al botón presionado.

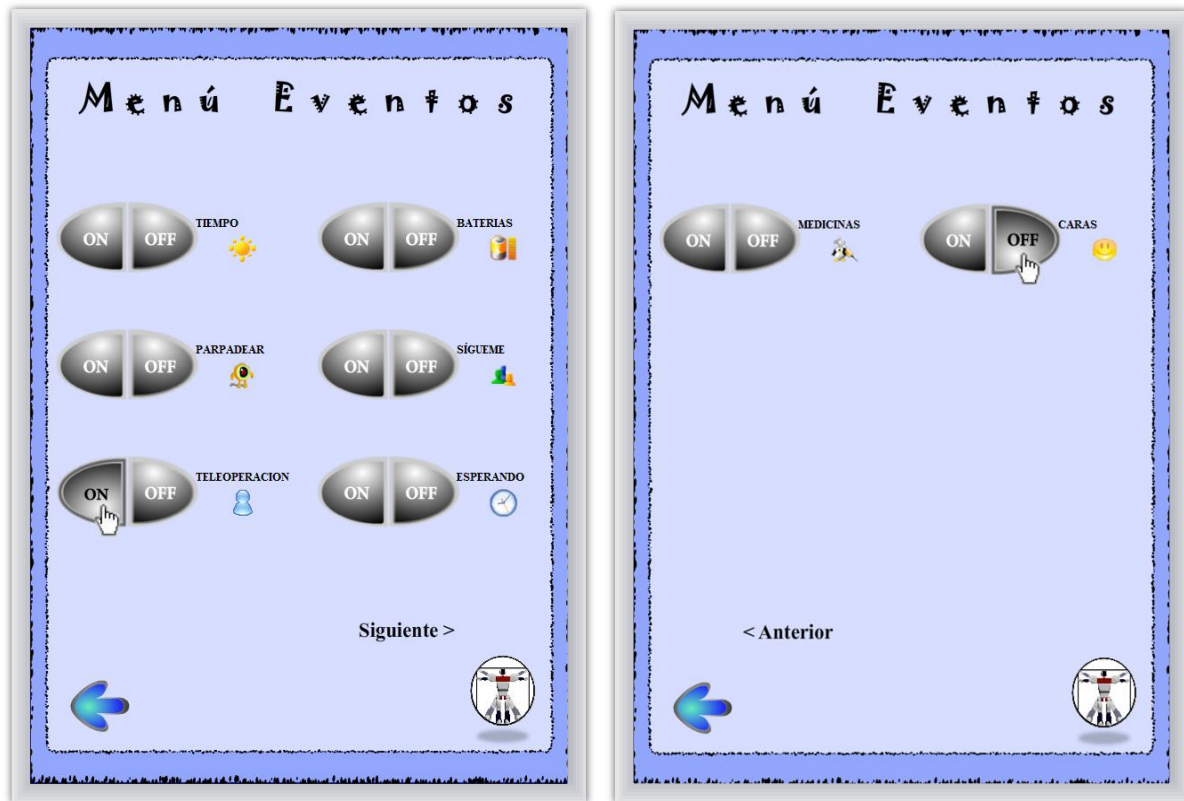


Figura 16. Interfaz del menú de eventos. Modelo de botón partido.

Debido a que la cantidad de habilidades no es conocida a priori, y es posible que todos ellos no puedan representarse en una sola ventana, se ha incluido una navegación interna dentro de la sección, para recorrer toda la lista. Esa es la función de los botones *Siguiente >* y *< Anterior*.

El diseño final contempla un máximo de seis elementos por pantalla. Si hay más, aparecerá un botón con el texto *Siguiente >*, de forma que al pulsar sobre él, se pase a los siguientes elementos de la lista, como si se hubiera pasado de página. Para tener una navegación completa, en páginas posteriores, aparecerá el botón con el texto *< Anterior*.

Además de estos elementos, también se puede acceder a la presentación de Maggie pulsando en el logo del laboratorio, o volver al menú principal pulsando la flecha azul.

4.4 Display de Estados

Como ya se comentó con anterioridad, este es un diseño preliminar de lo que en un futuro se mostrará en esta sección.

En la Figura 17 se muestran dos ejes con una barra de colores. Estas barras sirven para mostrar de forma gráfica, valores positivos y negativos dentro de un rango determinado. Una vez completo, servirá para representar un conjunto de valores, o estados representativos de Maggie.

La barra está compuesta por un degradado de colores, que dependerá de su tamaño, para añadirle un componente más visual y atractivo.

El tamaño es algo reducido respecto al espacio libre que queda, pero esto no se ha modificado debido a que en un futuro se espera que haya varios de estos gráficos en una misma ventana, representando distintos valores relacionados con Maggie. Por lo que se mantiene un tamaño adecuado para que puedan ajustarse, al menos, cuatro gráficos a la vez en la interfaz.

También aparecen los elementos interactivos comunes a todas las secciones de:

- La flecha de regreso, para volver al menú principal.
- El símbolo del laboratorio, para dar inicio a la presentación.

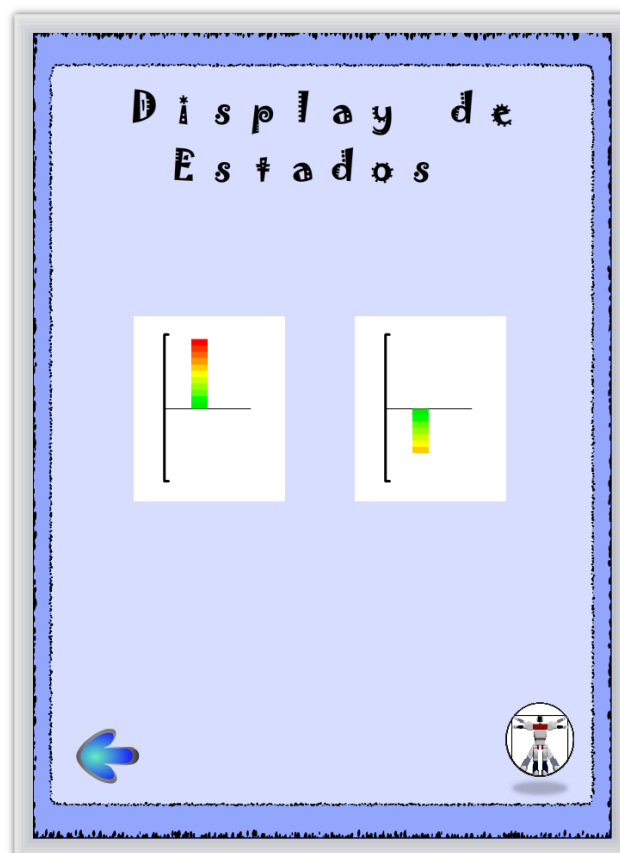


Figura 17. Interfaz del display de estados.

4.5 Menú de Juegos

En esta sección (ver Figura 18), se incluye un menú con la lista de juegos disponibles actualmente en la interfaz. El diseño en un comienzo era diferente del ideado para el menú principal, más vistoso, con otro fondo y utilizando otros botones con animaciones distintas. Sin embargo, finalmente se decidió mantener un estilo más uniforme, utilizando unas formas y colores homogéneos en las secciones, por lo que la versión final es ésta.

Debido a que realizar aplicaciones de entretenimiento dentro del programa, no era un objetivo principal del proyecto, se han incluido sólo dos juegos. De este modo, al haber menos elementos, se ha dado un poco más de espacio entre ellos del que hay en el menú principal. Para acceder a la ventana de inicio de cualquiera de las aplicaciones, se debe pulsar en el botón que se encuentra a la izquierda del nombre del juego.

Los demás elementos interactivos de la imagen son:

- La flecha para volver al menú principal.
- El icono del laboratorio, que abre la presentación.



Figura 18. Interfaz del menú de juegos.

4.5.1 Juego Bubbles

Este es un sencillo juego, que consiste en “explotar” las estrellas, como si fueran burbujas.

El inicio del juego muestra un espacio estrellado en movimiento, el nombre y un saludo de bienvenida, junto con una pequeña explicación sobre cómo jugar (ver Figura 19). Además de la flecha y el símbolo del laboratorio, hay una animación en el centro de la pantalla, como una estrella parpadeando. Al pulsar sobre ella, es cuando se da comienzo al juego, cambiando los elementos de la pantalla a como se ve en la Figura 20.

La interfaz del juego en sí, es muy sencilla respetando el diseño de la versión original, realizada por el personal del laboratorio en Flash 8 y ActionScript 1.0 (J. Gorostiza). Contiene (además de la flecha de retorno, y el icono del laboratorio), los siguientes elementos:

- **Las estrellas:** Aparecen y se mueven en la ventana de forma aleatoria. Al pulsar sobre una de ellas una vez, cambia de color, para simular que ha sido “tocada”. Al segundo toque, se desvanece.
- **Contador:** Arriba a la izquierda, aparece el contador del número de estrellas que hay en la pantalla y se deben de “explotar”. Según aparecen más estrellas, y el usuario hace que otras se desvanezcan, el contador va variando.



Figura 19. Interfaz de inicio del juego Bubbles.



Figura 20. Interfaz dentro y al final del juego *Bubbles*.

Cuando se acaban con todas las estrellas (esta cantidad es configurable a través del código) aparece la pantalla de fin de juego. Esta se ha incluido para no cortar de forma brusca el juego, regresando al menú directamente. De esta forma, se pasa primero a una ventana que informa de que la aplicación ha acabado, y se deja que sea el usuario el que regrese al menú de juegos, una vez que pulse sobre la flecha azul situada en la esquina inferior izquierda.

4.5.2 Juego Sapientino

Este es un juego que se quiso añadir al proyecto como aportación personal a Maggie. La idea era la de que fuera algo educativo, sin dejar de ser algo sencillo y vistoso para su uso por niños.

Sapientino, es un juego que de forma general, consiste en unir diversos conceptos. En nuestro caso, se trata de unir nombres de flora y fauna, típicos de un hábitat correspondiente a una laguna, con el elemento que se corresponda de la imagen que se ofrece (ver Figura 21).

Primero, se selecciona un nombre pulsando sobre el botón (ej.: sapo), y después, se debe pulsar sobre la zona de la imagen en la que éste animal o planta, se encuentre representado.



Figura 21. Interfaz de inicio y vista general del juego *Sapientino*.

En la pantalla de inicio, se incluye el nombre del juego, una breve explicación de lo qué hay que hacer, y un botón que debe pulsarse para empezar a jugar. Una vez dentro del juego, hay diversos elementos que requieren de mayor explicación:

- **Botones identificativos:** Cada botón contiene un nombre de un elemento de flora o fauna que aparece en algún punto de la imagen. Se han incluido nueve en total: *sapo*, *espadaña*, *pato*, *culebra*, *nenúfar*, *rana*, *renacuajo*, *caballito del diablo* y *zapatero*. Al pulsar sobre uno de estos botones, se comienza la ronda, de forma que los demás botones se desactivan y adquieren un cierto nivel de transparencia. A partir de ahí se

ha de jugar necesariamente con el nombre elegido hasta que se acierte a pulsar sobre la imagen adecuada.

Cuando se descubra (ver Figura 22), el botón pasará a estar desactivo durante el resto del juego, para diferenciar fácilmente los elementos que aún faltan por identificar, de los que ya se han hallado, y se podrá elegir otro botón para comenzar la siguiente ronda.

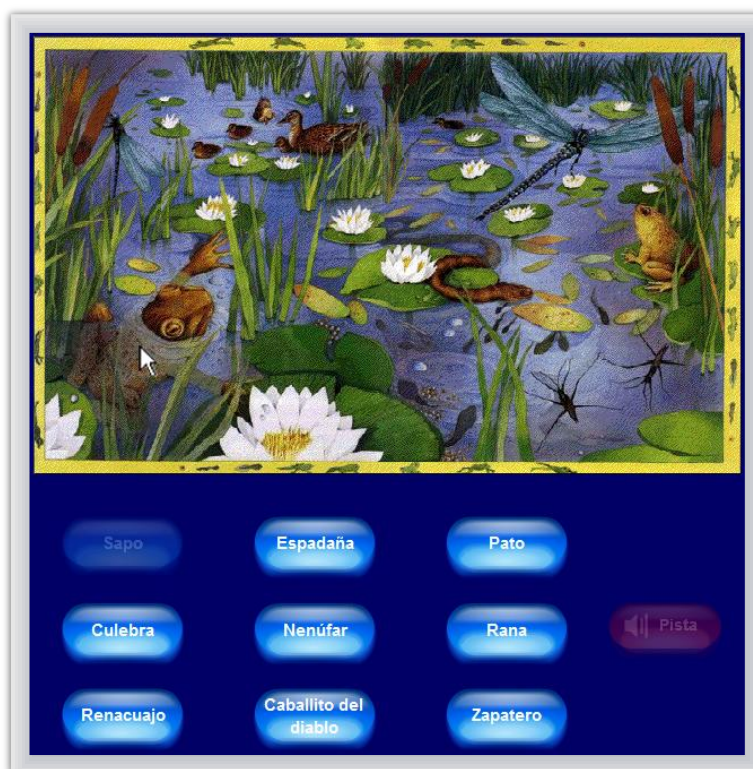


Figura 22. Interfaz al acertar una asociación del juego *Sapientino*.

- **Imagen de la laguna:** Esta es una imagen que permanece inactiva, hasta que se pulsa alguno de los botones del conjunto de nombres. Esto es así para que no se puedan obtener pistas sobre qué elementos de la imagen se pueden escoger, hasta que se elige el primer habitante de la laguna con el que jugar.

Si ya se ha elegido uno (por ej. el sapo), al pasar el ratón sobre la imagen, habrá distintas zonas que se iluminarán (ver Figura 23), mostrando el contorno de los elementos seleccionables. Cuando se hace clic en una zona iluminada, si se ha acertado en la unión de nombre-imagen, esa parte dejará de ser activa en las siguientes rondas.



Figura 23. Ejemplo de selección de un elemento en el *Sapientino*.

- **Las pistas:** El botón de pistas, solo aparecerá activo cuando se haya pulsado sobre uno de los botones con nombres. Si se tienen dudas sobre que elemento de la imagen corresponde con el nombre con el que se está jugando, se puede pedir una pista, pulsando este botón. De esta forma, será Maggie quien ofrezca por voz, una breve definición del animal o planta que se esté buscando, para que sea más fácil de identificar en la imagen.

Cuando termina el juego, una vez emparejados todos los nombres, con su imagen correspondiente, aparece una pantalla con los resultados obtenidos (ver Figura 24). Se muestran los porcentajes de aciertos y fallos en las respuestas dadas, mientras que Maggie será la que realice algún comentario por voz, sobre la participación del usuario en el juego.

Los elementos de la flecha para volver al menú de juegos, y la presentación de Maggie, están presentes durante todo el juego.



Figura 24. Interfaz de resultados de *Sapientino*.

4.6 Galería de Imágenes

En esta sección, a partir de un XML, se carga una lista variable de imágenes en miniatura, en la parte inferior de la pantalla. Si se pulsa con el ratón sobre cualquiera de ellas, aparecerá la imagen ampliada en la mitad superior de la interfaz (ver Figura 25).

Se han colocado de forma que aparezca un máximo de ocho fotos a la vez, ordenadas en dos filas de cuatro elementos cada una. En caso de que haya más de ocho elementos, se incluye una navegación interna en la sección, utilizando los botones de *Siguiente* > y < *Anterior*. Al pulsar en estos botones, se cargarán las siguientes imágenes, desapareciendo lo que hubiera antes cargado en la pantalla.

De cara a que la galería de imágenes fuera más atractiva, se han añadido algunas animaciones, como el que las miniaturas se vuelvan más transparentes, al pasar el ratón por encima de ellas, o que la carga de las ampliaciones tenga un efecto de degradado de blanco, hasta la completa aparición de la imagen seleccionada.



Figura 25. Interfaz de la galería de imágenes.

En esta sección, como en las demás, aparece la flecha para regresar al menú principal, y el logo del laboratorio, para abrir la presentación.

4.7 Reproductor de Música

En esta sección se ha implementado una lista de reproducción de música (ver Figura 26). Las canciones se cargan desde un XML (para ver más detalles ir al punto 6.3.6 Cómo Usar/Modificar la Lista de Reproducción de Música), y se han incluido todos los elementos básicos y funcionalidades de cualquier aplicación similar.



Figura 26. Interfaz del reproductor de música.

Al entrar en esta sección, se comienza a escuchar la primera canción de la lista. El título de la canción y el nombre del artista, aparecen resaltados en la parte superior de la interfaz. Por debajo de estos, aparece el control del tiempo, con un contador que va aumentando según el tiempo que ya ha pasado de reproducción, y otro que muestra la duración total de la canción.

El resto de elementos, tienen cierto nivel de interactividad, por lo que pasaremos a describirlos a continuación:

- **Botones de control:** Estos son los botones típicos de play, pausa, stop y para pasar a la canción siguiente o anterior, con una diseño que trata de dar una sensación de profundidad visual en estático, y al pulsarlos. Su función en cada caso está clara. Lo único a destacar, es que el botón de *play* y *pausa* nunca están activos al mismo tiempo. Cuando está sonando una pista de audio, el *play* estará desactivado. Al pausar, volverá a activarse, y se desactivará el botón *pausa*. En el caso de querer pasar de canción, también se irá modificando, la canción que está resaltada en la lista.
- **Control de audio:** Este botón se puede utilizar de dos formas:
 - **Mute:** Si se quiere silenciar una canción de forma inmediata, basta con pulsar el botón de audio una vez. Para volver al nivel de volumen estándar rápidamente, bastará con pulsar nuevamente este botón.
 - **Subir/bajar volumen:** Si se desea subir o bajar un poco el volumen y ajustarlo de forma más cuidadosa, al pasar con el ratón sobre el icono del audio, se verá como se despliega la barra del volumen hacia arriba (ver Figura 26). Si con la barra desplegada, se pulsa en algún punto de la misma, se ajustará el volumen según la posición donde se haya producido ese clic de ratón. Si el puntero se aleja de esta zona, la barra de volumen se replegará sobre el botón de audio, desapareciendo de la interfaz.
- **Barra:** Según va avanzando la reproducción de una canción, el puntero de la barra se irá moviendo de forma proporcional. En caso de que un usuario, quiera ir a un punto concreto de la pista de audio, podrá pinchar sobre la barra y arrastrar el puntero para desplazarse hacia delante o hacia atrás en la canción. El control de tiempo se modificará a la vez que se mueve la barra. Al soltar el puntero, la reproducción continuará desde donde el usuario haya marcado. Si la canción estaba en pausa, habrá que pulsar el *play* para que siga la reproducción.
- **Lista de canciones:** Esta lista muestra todas las canciones cargadas del XML. Se han modificado los colores, la forma y el tipo de letra del componente clásico, además de añadir un icono, y para darle una apariencia más atractiva.

De cara a la funcionalidad, si se pulsa sobre cualquiera de los componentes de la lista, se comenzará la reproducción de la pista de audio seleccionada de forma inmediata. También se ha dotado de una barra deslizador vertical, en caso de que los elementos de la lista no quepan en el hueco reservado.

Respecto al resto de elementos, se debe resaltar que si se pulsa sobre el icono del laboratorio de robótica, se pausará automáticamente cualquier reproducción en curso. La flecha servirá para volver al menú principal.

4.8 Reproductor de Video

En esta sección se ha incluido un reproductor de video, con las funciones básicas de control y audio, que actualmente contiene una película de dibujos animados (siempre teniendo en mente su uso con niños, aunque esto también puede ser de utilidad para mostrar videos relativos a Maggie, y al laboratorio de robótica).

En la Figura 27 se puede ver el diseño de la interfaz. Se ha incluido un botón de play/pausa, el stop, un botón específico para silenciar el audio del video, además de una barra, para ajustar el volumen con mayor exactitud. También se dispone de una barra cuyo puntero se puede arrastrar para desplazar la película hacia delante o hacia atrás.

El diseño de esta interfaz es más sencillo que el del reproductor de audio, debido a que en Flash ya existe un componente con apariencia fácilmente configurable para los videos, pero no para el audio. Por ello, la sección descrita en el punto 4.7 Reproductor de Música, es un diseño creado desde cero, para esta aplicación.



Figura 27. Interfaz del reproductor de video

5 Arquitectura del Programa

En este capítulo se detallará cual es la arquitectura del robot para el cual se ha hecho la aplicación, cómo se ha integrado el nuevo software en la arquitectura ya existente, cómo se ha realizado la comunicación entre los ordenadores (VisMaggie y el tabletPC) incluyendo una descripción del código implementado, y cuál es la arquitectura de ese nuevo software.

5.1 Arquitectura de Maggie

Para empezar es necesario explicar los rasgos generales de la arquitectura de Maggie, cómo funciona su software, así como resaltar una serie de puntos que serán de vital importancia para la correcta comprensión del programa diseñado.

En el punto 2.1.2 Maggie, ya se ha realizado un primer boceto de Maggie, explicando el objetivo de su investigación y el porqué de su diseño. Ahora pasaremos al cómo funciona y qué componentes tiene.

Maggie es un robot social que externamente, tiene una apariencia atractiva y amigable, que recuerda a un juguete de niños, con un cierto nivel de expresividad tanto verbal, como no verbal.

Internamente, a nivel de hardware se le ha dotado de las siguientes características:

- Movilidad con una base con ruedas y guiado diferencial.
- Distintos mecanismos de interacción, mediante:
 - Doce sensores capacitivos para la detección del tacto integrados en diferentes partes de la carcasa del robot, como cabeza, espalda o manos, entre otros. Esto sirve para dar la sensación de que Maggie está dotada de una piel sensible.
 - Cámara con procesamiento de imagen situada en la boca.
 - Dos antenas de radiofrecuencia.
 - Altavoces y un micrófono bluetooth para obtener información vocal del usuario (Gorostiza y cols., 2006).
 - Doce sensores infra-rojos, que permiten interactuar con elementos como una televisión o un equipo de música.
 - Doce sensores de ultrasonidos y un sensor láser para detectar cercanía de objetos.
 - Una cabeza móvil con dos grados de libertad, con párpados controlables, y leds azules en la boca que se iluminan en sincronía con la voz.
 - Dos brazos móviles, con un grado de libertad.

Un punto importante a tener en cuenta, es que para acceder a los elementos hardware de la arquitectura, se han desarrollado una serie de servidores (uno para cada dispositivo), encargados de mandar y recibir información desde todo el hardware del robot, usando la tecnología Remote Procedure Call (RPC). Este lenguaje es independiente de la arquitectura y el lenguaje de programación, por lo que junto al diseño realizado, permite un acceso distribuido cliente-servidor con varios clientes (que pueden estar en distintas máquinas con arquitecturas diferentes), realizando múltiples accesos al mismo recurso.

Pero es a nivel de software lo que más interesa en este trabajo.

Maggie tiene dos ordenadores (VisMaggie y un tabletPC), interconectados entre sí a través de una red Ethernet, y con acceso al exterior mediante WiFi 802.11. El tabletPC contiene la interfaz de usuario (que se explica en más detalle en el punto 5.3 Arquitectura del Programa Flash). Vismaggie, contiene todo el software de procesamiento de habilidades, y control de los diversos elementos físicos del robot y es esta última la que se detalla a continuación.

La arquitectura de software del robot Maggie está basada una arquitectura híbrida Automático-Deliberativa (Barber, 2000) (Salichs & Barber, 2001), que en adelante llamaré AD (ver Figura 28).

El nivel deliberativo agrupa los procesos de alto nivel con capacidad de razonamiento, y el nivel automático se asocia a los procesos que interactúan con actuadores y sensores del sistema, y que requieren de respuestas inmediatas ante la información recibida. La clave es que el nivel automático cumpla el objetivo si puede, y si no, pase al nivel superior, que es capaz de adaptarse y “descomponer” el problema en otros más sencillos que se ejecuten de forma secuencial.

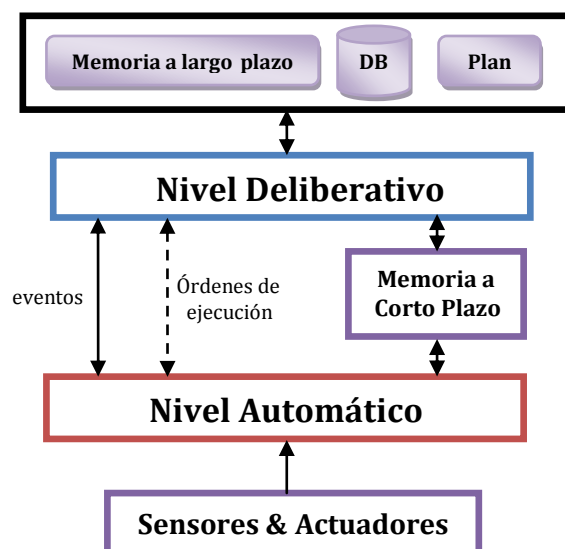


Figura 28. Arquitectura Automática-Deliberativa (AD)

El elemento básico de ambos niveles de la arquitectura AD se denomina *habilidad*. Una habilidad es la capacidad del robot de efectuar un razonamiento, de procesar información o realizar acciones. Se pueden activar mediante un secuenciador de órdenes o a partir de otra habilidad. Interactúan entre ellas mediante eventos, pudiendo enviar o recibir datos con una comunicación bidireccional entre los niveles automático y deliberativo, a través de la memoria a corto plazo. Esta memoria guarda información relevante obtenida a través de los sensores o facilitada por habilidades, es de rápido acceso pero con una capacidad más limitada y se pierde al terminar el ciclo de funcionamiento de Maggie. En cambio, la memoria a largo plazo es sólo accesible a través del nivel deliberativo, y sirve para aquellos datos que signifiquen un aprendizaje relevante por parte de Maggie, debiéndose almacenarse de forma perdurable, aunque con mayor lentitud de respuesta.

La memoria de Maggie sigue el paradigma de pizarra, según el cual, cuando un dato es guardado en memoria, este permanece disponible para su acceso por cualquier componente de la arquitectura que lo necesite.

De cara a los eventos, en la arquitectura AD son mensajes asíncronos que siguen el paradigma de publisher/subscriber. Cuando ocurre un suceso destacado en el sistema, se envía un evento que es recibido por todos los que se hayan suscrito a ese evento, que actuarán en consecuencia. El emisor puede asociar un número entero al evento, y enviarlo conjuntamente. El receptor tendrá que definir un método manejador del evento donde indique el proceso a realizar cuando reciba el evento.

A nivel de programación, también es importante entender el funcionamiento de las habilidades, eventos y memoria a corto plazo. Por ello se han incluido los siguientes ejemplos:

Las *habilidades* deben de heredar de la clase *CHabilidad*. Debido a la herencia, se tiene que sobrescribir el método abstracto *proceso()* de esta clase y se debe de incluir un constructor que tenga una cabecera similar a esta:

```
ClaseDerivada:: ClaseDerivada (unsigned long t): CHabilidad (t) { ... }
```

Cuando se crea un objeto de una clase derivada de *CHabilidad*, se ejecutará automáticamente, y de forma cíclica cada cierto tiempo *t* (en milisegundos), el método *proceso()* que se haya implementado. Este método es donde reside la principal funcionalidad de la clase.

Para los *eventos*, se deberá instanciar un objeto de tipo *CEventManager*. Para enviar un evento cuyo número entero identificador unívoco fuera *ID_EVENTO*, sería:

```
CEventManager gestorEventos;  
gestorEventos.emit(ID_EVENTO);
```

Para suscribirse a un evento, se haría de la siguiente forma:

```
gestorEventos.subscribe (ID_EVENTO,nombreFuncionManejadora,(void*)this);
```

Para dejar de estar suscrito (conviene hacer esto en el destructor de la clase):

```
gestorEventos.unsubscribe (ID_EVENTO);
```

La mayor parte de las habilidades se activan y bloquean mediante eventos para hacer que la arquitectura sea desacoplada. Los estados más utilizados son:

- LISTO: la habilidad está creada y espera poder ser ejecutada.
- BLOQUEADO: la habilidad está pausada.
- EJECUCIÓN: la habilidad está ejecutando.

Para que la habilidad funcione de esta forma se tendría que suscribir al evento de activación, y al de bloqueo, y luego gestionar esto en la función manejadora del evento. Para el caso del bloqueo podría ser algo parecido a:

```
void manejadorEventoBloqueo(void* aux, int p){
    ClaseHabilidad* pHab= (ClaseHabilidad *)aux;
    if (pHab->verEstado()== habAD::EJECUCION)
        pHab->bloquea();
}
```

Los métodos *bloquea()* y *activa()* que se usan para bloquear/activar la habilidad se heredan de *CHabilidad* por lo que no son funciones que se tengan que implementar aparte.

La memoria a corto plazo, es otro componente que se usa con frecuencia. Para guardar o leer un dato en memoria, se debe conocer cuál es el número entero que identifica de forma única ese dato y su tamaño. Cada uno de estos elementos, será un objeto de la clase *CItem*, y para acceder a él, será mediante los métodos de la clase *CmemCortoPlazo*. Un ejemplo de lectura de una cadena de 50 caracteres, con identificador *ID_DATO*, se haría:

```
char string[50];
CmemCortoPlazo mem;
CItem item(50);
item.asignarID(ID_DATO);
mem.ObtenerDatoActual(item);
item.obtenerDato(string); //se vuelca el contenido en la variable
```

Si se quisiera sobrescribir el dato anterior, habría que cambiar las dos últimas líneas por:

```
item.asignarDato((char*)&string, 50);
mem.ColocarDato(item);
```

Para más detalles y ejemplos del código, visitar la docuwiki ⁽¹¹⁾ del laboratorio.

5.2 Comunicación TabletPC-VisMaggie

El principal objetivo del proyecto, era conectar de alguna manera el tabletPC, con el ordenador que contiene toda la arquitectura AD de Maggie (VisMaggie), y crear una comunicación bidireccional entre ambos. Por un lado se diseñaría una interfaz gráfica, mientras que en VisMaggie habría que añadir una habilidad nueva que hiciera de conexión puente con el resto de la arquitectura, y lo que se pretendía era que la interfaz no fuera independiente, sino que interactuase con las habilidades de Maggie y viceversa.

Después de analizar cómo realizar la comunicación, de forma que las aplicaciones a desarrollar en Flash y C++, se integraran a la perfección con la arquitectura definida para Maggie, se llegó a la conclusión que lo más sencillo sería realizar una comunicación mediante sockets, con una arquitectura *cliente-servidor*. Esto garantizaría un modo de conexión con mensajes de texto asíncronos, sobre los que se podría crear un sencillo protocolo para analizar la información enviada rápidamente.

Respecto a los motivos principales para escoger el uso de sockets, además de cumplir con los requisitos básicos (como la independencia de los sistemas operativos o del lenguaje de los programas que hacen de cliente o servidor), los sockets son de fácil aplicación, y el grupo de trabajo del laboratorio ya estaba familiarizado con esta técnica. Esto era muy importante para no agudizar la curva de aprendizaje del nuevo software, ya que este trabajo no es concluyente, sino que representa una versión inicial susceptible de mejoras. Además, debido a que la concurrencia de la comunicación es casi nula, y de momento no se prevé que haya más de un cliente conectado a la vez, no era necesario emplear métodos más potentes.

El servidor se encuentra contenido en VisMaggie y el programa cliente es la aplicación Flash. Esto se hizo así debido a que las librerías de socket que utiliza ActionScript son limitadas y están preparadas para hacer aplicaciones cliente de servidores web.

Por lo tanto, realizar la parte servidor junto con la interfaz Flash, implicaría crear un programa a parte en C, C++, u otro lenguaje, que realizara el control de los sockets como servidor, y se comunicara con dos clientes: uno en VisMaggie y otro que fuera la aplicación Flash. Esta se consideró una solución más complicada, y menos intuitiva, que unir el servidor con la habilidad de Maggie, realizándolo todo en C++, manteniendo un solo canal de comunicación a un cliente y no a varios.

Tras una serie de reuniones con el personal del laboratorio, se decidió que la parte contenida en Vismaggie tuviera una estructura que separase, claramente, la parte que contenía la funcionalidad de los sockets, y la que realizase las tareas de comunicación con la arquitectura de Maggie, de forma que enviar o recibir información del socket, fuera transparente para el usuario. Estas decisiones de diseño, se tomaron para hacer una organización clara, separando el código según la funcionalidad, para que fuera fácilmente comprensible por otros técnicos que lo quieran utilizar o modificar, y para que a su vez, el programa servidor pudiera utilizarse separado de la habilidad, asociándolo a otra aplicación, en caso de necesidad, en un futuro.

Debido a los cambios necesarios para que el código se fuera ajustando a las decisiones de diseño tomadas, hubo varias versiones diferentes, donde se fue puliendo la estructura óptima de la arquitectura, hasta llegar a la definitiva que se puede ver la siguiente figura.

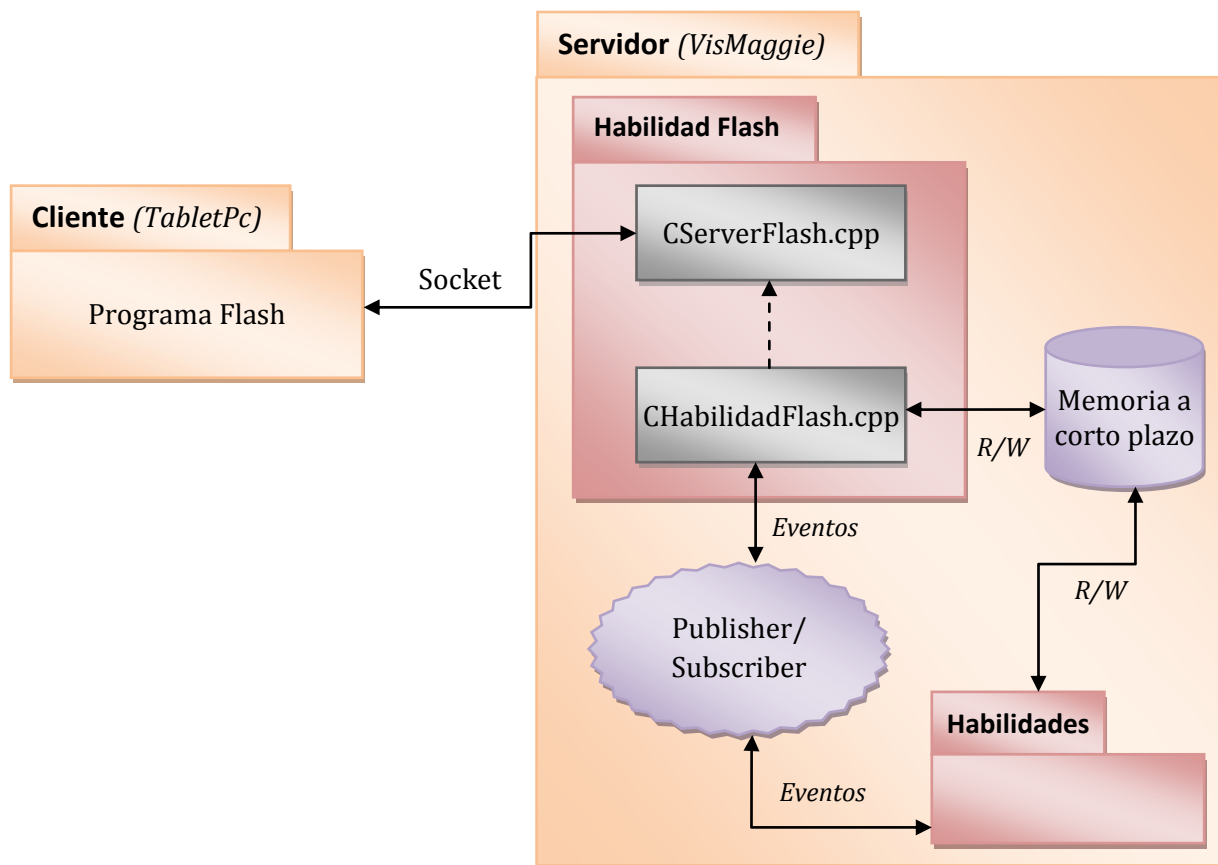


Figura 29. Arquitectura cliente-servidor.

Los ficheros fundamentales son *CServerFlash.cpp* y *CHabilidadFlash.cpp*. Como se puede deducir a partir de la Figura 29, el primero, es donde está implementado el servidor y el segundo, la habilidad que sirve de unión entre los componentes de la arquitectura AD y la comunicación por sockets.

A continuación se pasará a explicar en más detalle el funcionamiento de estos dos archivos:

- **CServerFlash:** Clase auxiliar que gestiona un servidor por sockets con el `tabletpc`. Una vez recibida la petición, se fija la conexión, y se lanza un hilo que esté siempre a la escucha para recibir mensajes del tablet, y también preparado para enviarlos. El socket se cierra al hacer la llamada al método destructor de la clase.

Se utilizan hilos y mutex, para asegurar que los mensajes sean totalmente asíncronos, y que la información que se obtenga del buffer, sea la correcta.

El tamaño de los mensajes se fija a cien caracteres en esta clase, en la constante `TAM_BUFFER`. En cuanto al número del puerto, se utiliza la constante `PORT`, con el valor cuatro mil, para facilitar modificaciones, sin tener que investigar en profundidad el código.

Las funciones principales de esta clase, son:

- `void CServerFlash::init():` Aquí se crea el socket, se asigna el número de puerto con la función `bind`, y queda a la escucha de que llegue una conexión de un cliente al puerto fijado con `listen`. Cuando se confirma una conexión con `accept`, lanza un hilo que estará esperando los mensajes entrantes, con la función estática `recv`.

```
pthread_create(&tabletpc, NULL, this->recv, (void*)this);
```

Esta función no es estática de forma arbitraria. La razón se debe a que C++, al ser un lenguaje de programación OO, el trabajo con sockets tiene más restricciones de las que tiene C. La principal a tener en cuenta, es que no se pueden crear hilos pasando funciones de clase que no estén definidas como `static`. Y dentro de un método estático, no se puede hacer referencia directa sobre los atributos de la clase. La palabra reservada `this`, sirve para obtener una referencia al objeto de la clase, pasarlo como parámetro de `recv`, y poder acceder (esta vez de forma indirecta), a los elementos que sean necesarios.

El hilo principal se ocupará de los mensajes salientes.

- `static void * CServerFlash::recv(void* aux):` Este es el método que se ejecuta en un hilo secundario, cuando se recoge la conexión del cliente. La ejecución queda bloqueada en un bucle, con la llamada a `read`, que lee los datos recibidos del socket, y lo guarda en `buffer_recibidos` (un atributo de la clase). Cuando ha llegado un mensaje, se marca el atributo `datosFlashRecibidos` a `true`, para que de un modo sencillo se pueda comprobar en la habilidad, si hay un mensaje esperando.

El parámetro *aux* que recibe esta función, se corresponde al código *(void*)this*, utilizado en la creación del hilo, de forma que es necesario realizar un casting a *CServerFlash* de *aux*, para poder utilizarlo correctamente.

- `void CServerFlash::getDatosFlash(char * datos):` La finalidad de esta función es la de guardar en el `char*` que le llega por parámetro lo contenido en el buffer de datos recibidos de la clase (*buffer_recibidos*), si se comprueba que se haya recibido algún mensaje que falte por leer. Esto es útil, para que en la habilidad se pueda obtener la información del buffer fácilmente y en el momento en que se vaya a utilizar. Se usan mutex para prevenir errores en caso de accesos simultáneos.
 - `void CServerFlash::send(char * datos):` Este método público, se emplea en la habilidad para enviar datos al `tabletpc`, de una forma transparente al usuario. Al realizar una llamada a esta función, se carga en el buffer *enviados* del socket lo contenido en el parámetro *datos*, y se proceder a mandar a la aplicación Flash utilizando la función *write*.
- **CHabilidadFlash:** Esta habilidad se encarga de gestionar la información recibida desde el `tabletpc`, avisando al resto de la arquitectura adscrita al evento *MENSAJE_RECIBIDO_FLASH*. También controla el envío de información al tablet, en caso de que alguna otra habilidad tenga información que mandar, y emita el evento *MENSAJE_LISTO_FLASH*. En el destructor de la clase se libera la memoria, y se eliminan los eventos.

Esta clase utiliza un objeto del tipo *CServerFlash*, para tener acceso a la funcionalidad de recibir y enviar mensajes. Los identificadores de los eventos, y de los ítems de memoria a corto plazo, también se incluyen en esta clase como constantes.

- **Constructor:** Cuando se crea un objeto de tipo *CHabilidadFlash*, se inicia el proceso de ejecución principal del programa. Para empezar, se inicializa el servidor, haciendo una llamada al método *init()*, de *CServerFlash*. Esto hace que se ponga en marcha el servidor, esperando las conexiones de sockets tal como se ha explicado con anterioridad. Una vez realizada la conexión, un hilo de ejecución quedará esperando los mensajes recibidos, y el principal, volverá al constructor de *CHabilidadFlash*, donde pasará a inicializar el objeto de control de la habilidad de habla de Maggie.

```
this->etts.activaEttsSkill();
```

Después de esto, se pasará al control de eventos. Para ello, se suscribe la habilidad a tres eventos diferentes (los identificadores son constantes de la clase):

- FLASH_START & FLASH_STOP: Se utiliza para gestionar el inicio/fin de la ejecución del método *proceso* de la habilidad, mediante eventos, al igual que se hace en el resto de la arquitectura AD.
- MENSAJE_LISTO_FLASH: Este evento sirve para advertir cuando otra habilidad de Maggie, pide enviar algún tipo de información al tabletPC.

Además de esto, se pasa a inicializar el atributo *hayDatos* a cero (será el indicador para detectar en las sucesivas ejecuciones del método *proceso()*, si se ha recibido el evento *MENSAJE_LISTO_FLASH*), y se registran los identificadores que se han de utilizar para manejar los ítems de memoria a los que se accederá. Esos identificadores son:

- DATA_TO_FLASH: Item de MCP, con la información que se ha de enviar a Flash.
- DATA_FROM_FLASH: Identificador del elemento de MCP, con los datos que se han recibido desde la interfaz en el tabletPC.

```
CItem item(TAM_SOCKET);
item.asignarID(DATA_TO_FLASH);
this->memCortoPlazo.RegistrarDato(item);
```

- Proceso: Método que se ejecuta de forma cíclica, cada vez que pasa el tiempo definido en el constructor de la clase. En este caso se utiliza para comprobar dos cosas:

- Si hay una petición de envío de datos al tablet, se manda la información. Esto se comprueba si el atributo *hayDatos* es *true*.
- Si se ha recibido información DESDE el tablet (cuando *datosFlashRecibidos* es *true*), se obtiene el mensaje mediante el método *getDatosFlash*, pasando el *char* datos* por referencia, de forma que lo que se copie del buffer al parámetro, dentro de la función, permanecerá al volver.

```
this->serverFlash.getDatosFlash((char*)&datos);
```

Una vez hecho esto, se analiza su formato, utilizando la función *strtok* para partir la variable *datos* en varios pedazos, utilizando de elemento separador el símbolo ":". El puntero que devuelve la

función se puede recorrer, como si fuera un array, para tener acceso a cada parte en la que se ha dividido la variable.

Las distintas cabeceras de los mensajes son:

- “ev”: Esto nos indica que se pide enviar un evento determinado a la arquitectura AD.
- “say”: Se emplea para hacer que Maggie diga algo en inglés.
- “di”: Su uso indica que Maggie tiene que decir algo en español.
- Otras: Si la cabecera del mensaje no corresponde con ninguna de las anteriores, el texto se guarda en memoria a corto plazo, en el elemento cuyo identificador corresponde a *DATA_FROM_FLASH*, y para avisar de la llegada de esos datos, se envía el evento *MENSAJE_RECIBIDO_FLASH*.

Si se quisiera añadir alguna más, sería cuestión de incluir otra sentencia *if else*, similar a las anteriores, con el caso particular.

- `manejadorEventoStart`: Si la habilidad estaba bloqueada o se acaba de iniciar, cambia su estado a *EJECUCIÓN*.
- `manejadorEventoStop`: Si la habilidad está activa, cambia a *BLOQUEADO*.
- `mensajeFlashPreparado`: Función manejadora del evento que se recibe, cuando desde otra habilidad se quiere enviar información al `tabletpc`. Esa información ya se habrá escrito en memoria a corto plazo, en un elemento con identificador *DATA_TO_FLASH*, que será una cadena de cien caracteres.

Existen tres casos en los que la información es directamente procesada por la habilidad que se acaba de detallar. El caso del control de eventos, se incluyó, ya que consta como requisito que la aplicación estuviera dotada de esa función. Para el caso del habla, se consideró necesario más tarde, debido a que se advirtió que iba a ser un mensaje muy utilizado en diferentes situaciones, por lo que lo más sencillo era incluir su tratamiento en esta habilidad, y no depender de otra secundaria, que solo dificultaría la comprensión del flujo del código. Sin embargo, el resto de mensajes serán almacenados en la posición correspondiente de memoria a corto plazo, y deberán ser las demás habilidades las que deban procesar como corresponda el texto del mensaje.

Se recomienda ver la Tabla 30. RNF-13. Protocolo de mensajes, para más detalle.

A estos se añade el mensaje para modificar el display de Estados, pero para ello ver el punto 6.3.4 Cómo Usar/Modificar el Display de Estados.

Para ver el flujo de datos de forma gráfica, se incluye el siguiente diagrama:

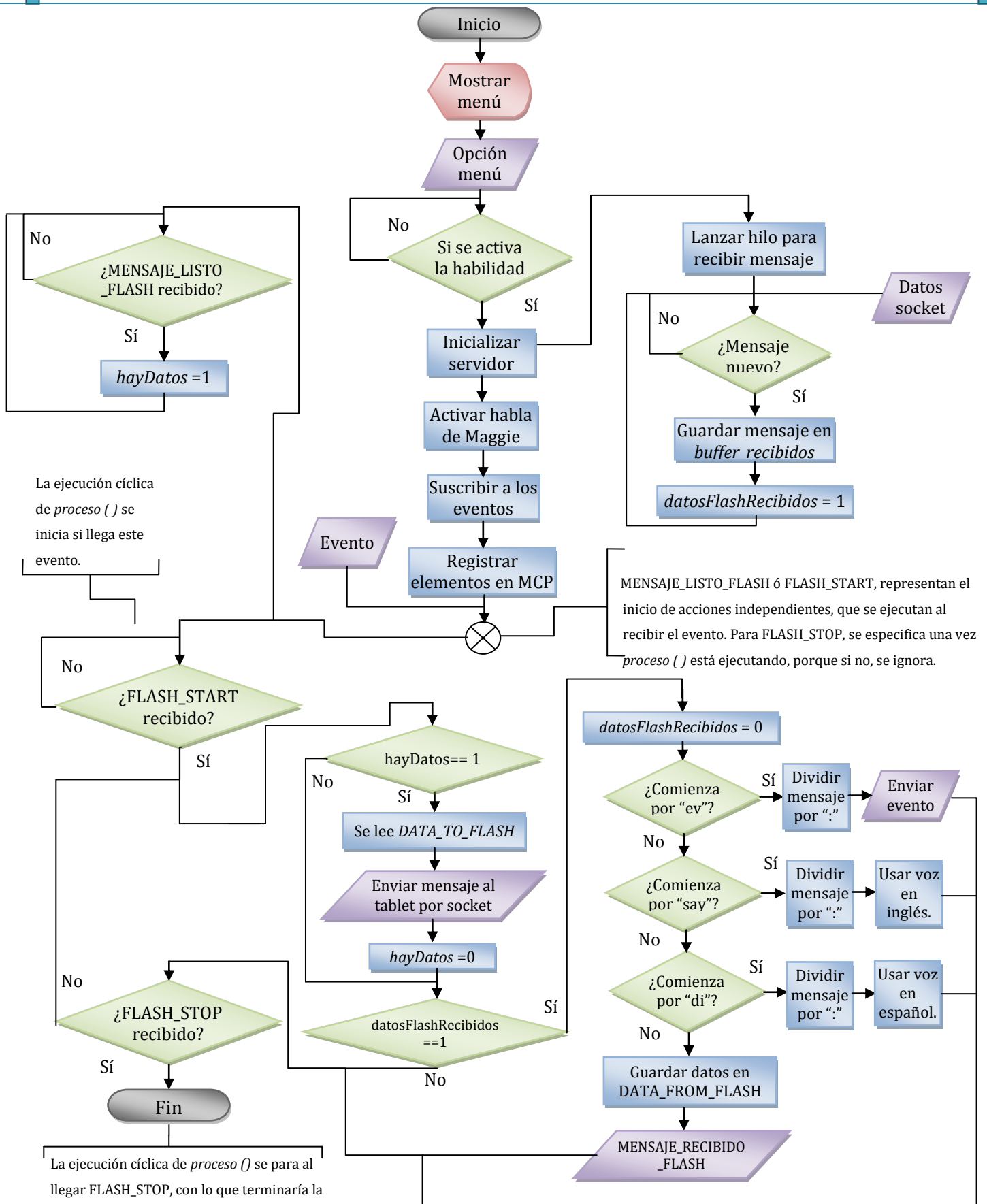


Figura 30. Diagrama de flujo de datos de la habilidad.

5.2.1 Seguridad

La seguridad fue un punto importante a tener en cuenta a la hora de programar toda la aplicación. Por un lado, los problemas de seguridad que plantea Flash, y las limitaciones a la hora de realizar una conexión debido a su sandbox, y por otro lado, las barreras que se han levantado en torno a VisMaggie, como protección ante posibles intrusiones, y de las que no se tenía un conocimiento al inicio del proyecto.

Debido a que Flash está principalmente orientado a aplicaciones cliente de servidores web, es necesario un sistema de seguridad que controle la transferencia de información que puede suponer un riesgo de seguridad o privacidad. Este sistema es a lo que se denomina sandbox. Su principal objetivo es la de asegurar la integridad de la máquina cliente, y de los archivos que se utilicen. Los archivos Flash ejecutan dentro del sandbox, y el acceso a la información que hay dentro del sandbox desde fuera, está muy limitado. Por lo que todos los archivos que se lean en local, unidos al cliente, no tienen problemas de acceso, pero sí cuando se accede a elementos externos. Esto se debe de tener en cuenta, en caso de querer cargar, directamente, una imagen desde una web de internet, ya que se accedería desde la ip del tablet a información que estaría fuera de su dominio, y se obtendría una violación de seguridad de sandbox (un cliente no puede acceder a información a un dominio distinto del suyo, y esto se controla por la ip, o la ruta web). En el caso de este proyecto, la interfaz Flash dispone de todos los elementos necesarios en local, guardados junto al ejecutable, de forma que no representa un problema.

Por la parte de VisMaggie, al ser un ordenador con información muy delicada, debido a que contiene toda la arquitectura AD de Maggie, tiene varias medidas de seguridad para impedir el libre acceso. Sin embargo, ya que hace el papel de servidor, era necesario abrir un puerto de VisMaggie que no estuviera en uso, para que se pudiera producir la conexión, ya que por defecto, todos están bloqueados ante conexiones entrantes. En un inicio, este bloqueo supuso un grave problema, debido a que se desconocía esa medida de seguridad, y fallaba reiteradamente la comunicación entre ambos PCs por ese motivo.

Una vez que se analizó lo que ocurría en las tablas de enrutamiento, y en los puertos de ambos ordenadores al realizar una petición de conexión, se descubrió el problema, y se realizó la petición de acceso al administrador de red de VisMaggie.

El puerto utilizado es el número 4000, el cual se comprobó previamente que estuviera libre, y su apertura requería de una serie de medidas de seguridad para que no se convirtiera en un riesgo de posibles accesos malintencionados.

De esta forma, sólo las peticiones que llegan desde la ip fija con la que está configurado el tabletPC son aceptadas. Esto se hizo añadiendo esa ip concreta como excepción de acceso al puerto deseado, en las iptables del sistema operativo Fedora de VisMaggie.

Además, el acceso de personal externo al tabletPC es complicado, ya que es un ordenador que se encuentra habitualmente desconectado, y cuando está encendido, sólo se puede acceder a él de forma remota, mediante el programa TightVNC. Esta aplicación se añadió durante este proyecto, para que ejecutase de forma automática al inicio del sistema operativo. Ofrece un entorno de servidor de escritorio remoto, accesible al ejecutar un cliente de TightVNC en otro ordenador, introduciendo la ip del tabletPC y la contraseña correspondiente con la que está protegida la comunicación. Previamente a esto, también se debe de conocer la contraseña del perfil de Windows del ordenador. Sin embargo, para poder acceder al tablet, se debe de realizar la petición desde otro ordenador que se encuentre conectado a la red local del laboratorio de robótica, ya que esta red está protegida por un cortafuegos que no admite conexiones externas, aunque se trate de otro ordenador conectado a otro segmento de la red general de la universidad. Tampoco sirve el acceso por escritorio remoto de Windows, ya que el tabletPC tiene instalado el Windows Vista Home Premium, y en general, los Windows Home no admiten conexiones entrantes de escritorio remoto.

Para conectarse a VisMaggie desde remoto, sirve realizar una petición SSH con Linux, o una aplicación que dé el mismo servicio desde un sistema operativo Windows, con un ordenador que se encuentre conectado dentro de la red del laboratorio. Para ello, se debe estar previamente registrado con un usuario en VisMaggie, y entrar con el identificador y la contraseña adecuados. A pesar de esto, para realizar modificaciones en Maggie fuera de los permisos básicos como usuario estándar, se debe conectar como root, opción protegida por contraseña.

Además de este usuario normal que se crea en Vismaggie, para poder acceder a la arquitectura, se ha de tener una rama creada para ese usuario en el repositorio de versiones del laboratorio. Ahí es donde realmente cada usuario puede acceder a una copia de la última versión estable de la arquitectura AD de Maggie, (ya que siempre se está trabajando en varias zonas a la vez, por distinto personal investigador, y cada uno trabaja en su rama para no sobrescribir en la de los demás, solo subiendo algo a la general, cuando ya es plenamente funcional). Una vez que se tenga una rama propia en el repositorio, se podrá bajar esta rama a Vismaggie, o a uno de los PCs del laboratorio, y se podrá trabajar con ella. Aun así, para poder utilizar alguno de los servidores de Maggie, como el de eventos, memoria a corto plazo o similar, será necesario subir estos eventos previamente, mediante un script "startAD.sh". Y estos scripts, sólo están disponibles conectándose como root.

En resumen Maggie está muy protegida por distintos niveles de usuarios, contraseñas, filtros de ips dentro del propio PC para salvaguardar los puertos de accesos no deseados, además de un cortafuegos del servidor para aquellos accesos de fuera de la red.

5.3 Arquitectura del Programa Flash

En este punto se detallará la estructura en alto nivel de la interfaz diseñada en Flash, para dar a conocer la relación entre las clases. Si se desea ver la explicación del funcionamiento de cada clase implementada en ActionScript 3.0 a más bajo nivel, ir a 6.3 Manual de la Aplicación.

La elección de utilizar Flash, fue un requisito al inicio del proyecto. A pesar de que los programas para el desarrollo de Flash son para Windows, a la hora de publicarlos (obtener el ejecutable), se puede hacer en diversas extensiones, como *.swf, *.html, *.exe, *.gif, o *.app. Esto implica que un mismo proyecto, se puede exportar en distintos formatos, y poder utilizarse en otros sistemas operativos además de Windows. Además, investigadores del laboratorio ya había utilizado Flash 8 en otros programas y juegos, por lo que en un principio la idea fue continuar con ese trabajo, y por eso las primeras versiones del proyecto fueron realizadas con Flash 8 y las versiones anteriores de ActionScript 1.0 y 2.0.

Sin embargo, una vez que la investigación y la experiencia sobre Flash fueron aumentando, se decidió cambiar todo lo hecho hasta la fecha y pasarlo a las versiones más modernas de Flash, en concreto la versión CS4, ya que la versión 8 había quedado hacia tiempo anticuada. Esto supuso un retraso en el tiempo global empleado en este trabajo, pero se consideró necesario para poder tener un código que perdurase más tiempo en uso. Se pasó todo el código a programación OO, con ActionScript 3.0. Colocándolo todo en paquetes con archivos por separado, se creó una estructura de clases que ayuda a tener mucho más organizado el código, a diferencia de las versiones antiguas, cuando todo se escribía dentro de los fotogramas del proyecto Flash, lo que aumentaba el peso del archivo a ejecutar y ralentizaba la aplicación.

El criterio para organizar el código ha sido sobre la base de la funcionalidad de cada parte, de forma análoga a lo ya visto en el punto 4. Diseño de la Interfaz de Usuario, dividido en secciones.

El código correspondiente a cada componente se ha guardado en paquetes por separado, además de los ficheros necesarios para el funcionamiento de cada uno de ellos, como imágenes, pistas de audio, videos y los archivos XML. El único que se diferencia del resto es el paquete “com”, que contiene los botones y animaciones que se han incluido mediante código en algún punto de la interfaz.

La clase Main es la principal del proyecto. Conecta las secciones, realiza las funciones de apertura y control del envío y recepción de los mensajes a través del socket, además de la navegación principal mediante los botones de la interfaz.

El siguiente diagrama de clases se ha incluido, para mostrar cual es la estructura de clases del proyecto Flash de una forma más visual.

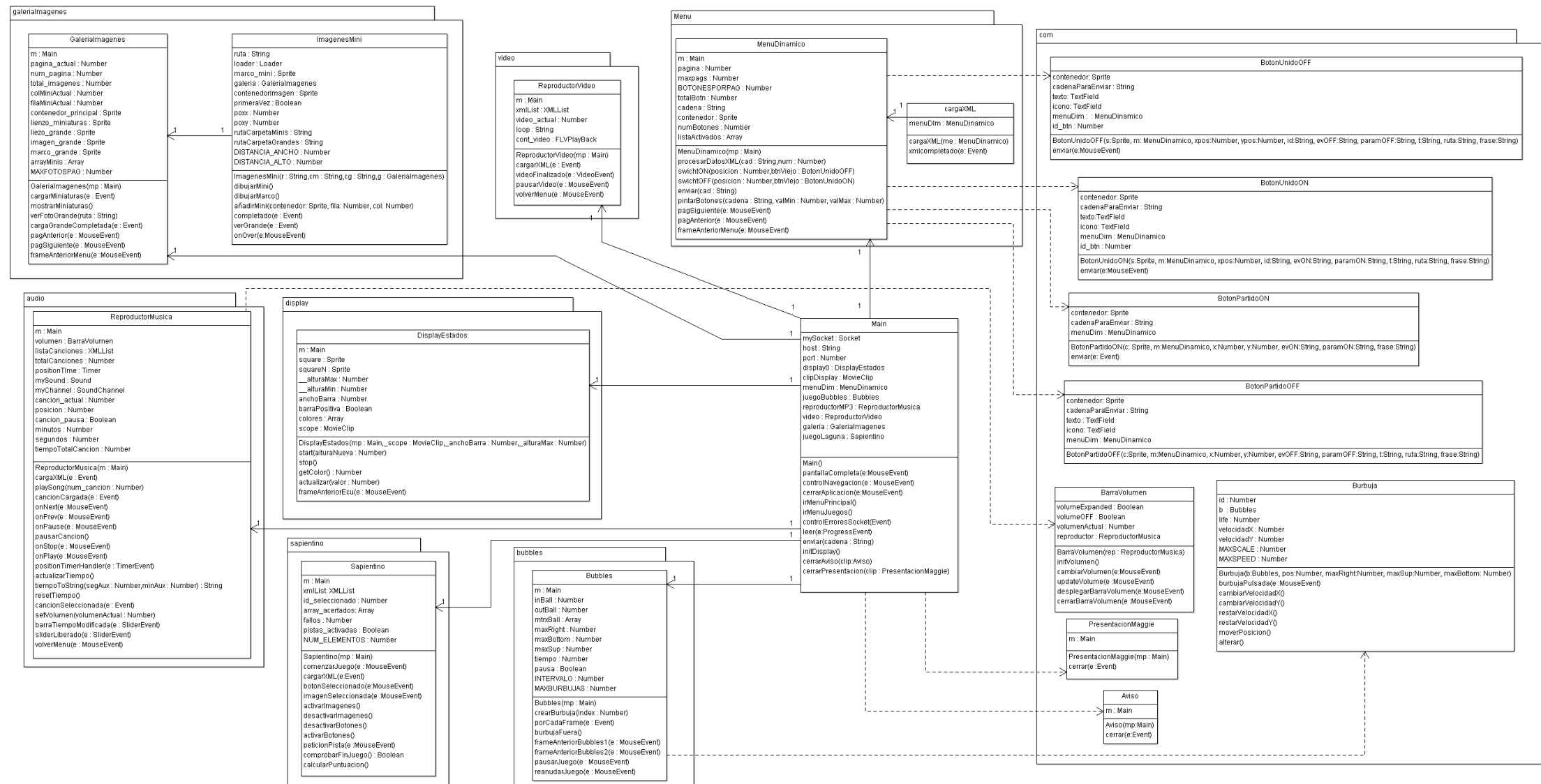


Figura 31. Diagrama de clases de la aplicación Flash.

6 Manuales

Una vez detallada en profundidad la arquitectura de la aplicación, en este capítulo se describirá distintos puntos a tener en cuenta a la hora de que otros desarrolladores quieran trabajar con la aplicación en un futuro. Las explicaciones se han hecho para personas sin conocimientos previos de Flash, pero si con cierta experiencia programando con lenguajes OO, de forma que seguirán un proceso de aprendizaje similar al que tuvo que seguir la autora de este proyecto, desde cero, hasta los resultados finales obtenidos.

Se han incluido tres manuales en lugar de uno, ya que se pensó que ésta sería la forma más organizada de poder explicar la aplicación apropiadamente. Esto se debe a que al detallar, por ejemplo, cómo se hace un botón, además de la función de ese botón en la interfaz, se debe exponer cómo se hizo la parte gráfica y cuál fue su manejo de eventos. Y ya que esto ocurriría con cualquier elemento de la interfaz, para evitar repeticiones, se decidió dividir el manual siguiendo esta idea. Por un lado se explicaría la parte gráfica. Por otro, las estructuras más utilizadas de ActionScript, necesarias para la comprensión del código. Y finalmente, la aplicación de ambas cosas para lograr los objetivos de cada sección.

En caso de ser necesario un mayor detalle, se aconseja leer los manuales oficiales de Flash Professional CS4 ⁽¹²⁾ y programación en ActionScript 3.0 ⁽¹³⁾ de Adobe.

6.1 Breve Manual Introductorio a Flash

En este punto, se explican unos conocimientos básicos sobre el manejo del software utilizado para realizar la interfaz, que es la versión Flash Professional CS4. Conceptos tales como que es un fotograma (frame), o de que se trata cuando se hace referencia al escenario, o la línea de tiempo, son convenientes conocerlos antes de empezar a trabajar con Flash, y también son necesarios para la correcta comprensión de los siguientes apartados del proyecto.

6.1.1 El Espacio de Trabajo y las Herramientas.

Se denomina *espacio de trabajo*, a la disposición de paneles, barras y ventanas de la aplicación, configurable seleccionando uno de los perfiles disponibles en la pestaña **Ventana > Espacio de trabajo**. Se puede personalizar, agrupando, acoplando o apilando los paneles.

En la Figura 32, se puede observar como es el programa en su perfil “Clásico”, y se hace referencia a los siguientes elementos:

- A. Barra de menús:** Éstos son los típicos menús de Archivo, Edición, Ver... que tienen la mayoría de los programas. El más notable puede ser “Control”, con controles sobre la animación, y para compilar y probar la escena actual o la película completa.
- B. Panel de muestras:** Este panel contiene una selección de las muestras de colores por defecto, y las últimas utilizadas, para poder realizar un acceso rápido a ellas.
- C. Paleta de colores:** Sirve para que el usuario busque el color deseado, eligiendo entre colores sólidos o degradados, variando el tono y nivel de transparencia.
- D. Conmutador de lugar de trabajo:** Despliega un menú para cambiar más rápidamente entre los distintos perfiles configurados.
- E. Botón contraer panel.**
- F. Inspector de propiedades:** Facilita información sobre los atributos más utilizados de la selección actual, ya sea de un objeto (color, posición, tamaño...), o del escenario (versión ActionScript y de reproductor Flash que se utiliza, y nombre de la clase principal asociada al escenario). Se puede modificar una propiedad del objeto desde aquí, sin acceder a los menús o paneles específicos.
- G. Panel Alinear:** Proporciona acceso directo a botones útiles para alinear objetos.
- H. Panel información:** Muestra la posición exacta del puntero del ratón al moverlo por la pantalla, y datos sobre la posición del objeto que esté seleccionado, si hay alguno.
- I. Panel transformar:** Acceso rápido para rotar y sesgar objetos, según los ejes x, y, z.
- J. Panel de herramientas:** Contiene utilidades de selección, modificación, dibujo, pintura, texto, etc. Las herramientas relacionadas están agrupadas (icono marcado con una marca en la esquina), como la transformación lineal, y la transformación de degradados. Manteniendo pulsado el ratón sobre el icono, se despliegan las opciones.
- K. Ventanas del documento.** Exhibe el archivo en el que se trabaja, y permite agrupar y acoplar ventanas de documento, en forma de fichas.
- L. Línea de tiempo.** Lista de capas de la escena, y de fotogramas.
- M. Panel biblioteca.** Aquí aparecen toda la lista de objetos creados en el proyecto.
- N. Configuración predefinida de movimiento.**

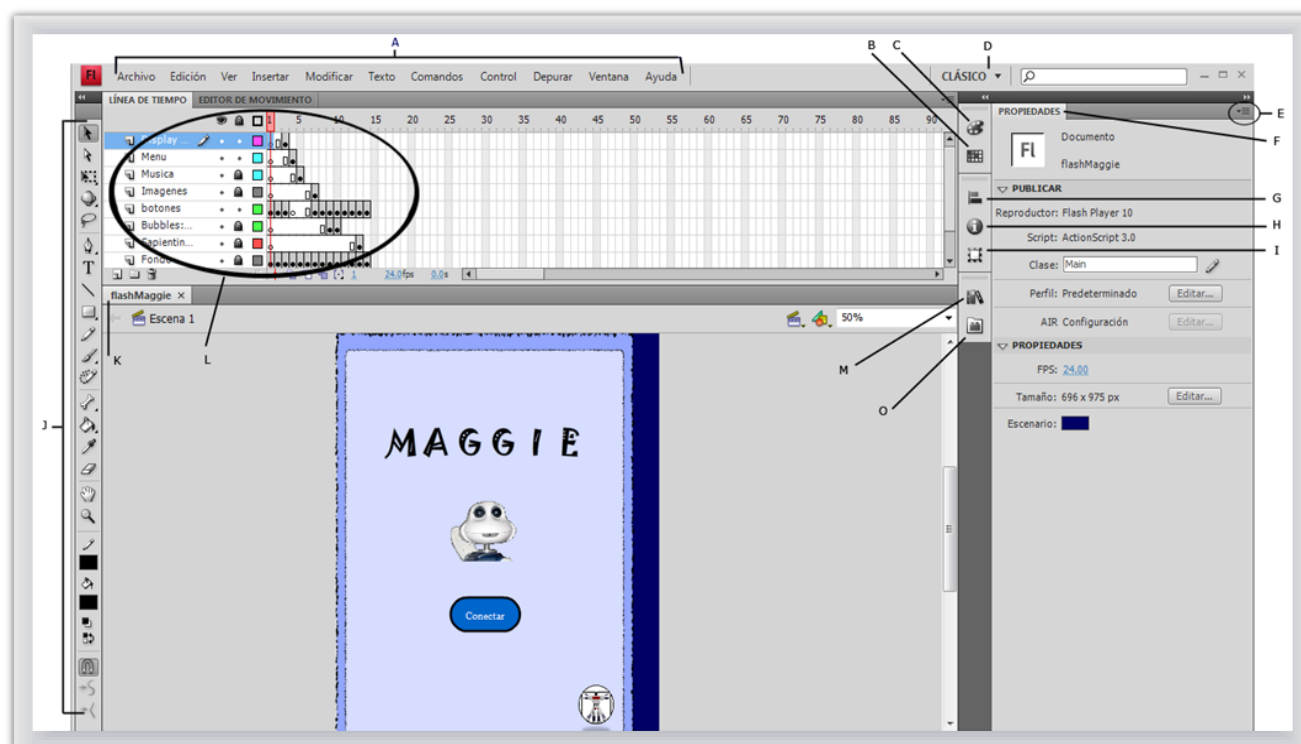


Figura 32. Espacio de trabajo con perfil "Clásico" de Flash CS4.

6.1.2 El Escenario

El *escenario* es el área donde se encuentra el contenido gráfico de los documentos Flash. Se puede cambiar el color de fondo, acercar y alejar, o incluir una cuadrícula, guías y reglas como ayuda a la hora de colocar elementos sobre él. Para que se abra el menú con estas opciones, se debe pulsar con el botón derecho del ratón en una zona vacía del mismo, mientras que los atributos principales, se mostrarán en el inspector de propiedades.

Los objetos que aparecen en el escenario, dependen en todo momento del fotograma y la escena seleccionada, que se definen como:

- Un *fotograma* es un instante de tiempo dentro de una película. Cuanto mayor sea el número de fotogramas, más durará la animación. Por separado son imágenes estáticas, que al mostrarse secuencialmente con una cierta velocidad, dan la sensación de un movimiento fluido al ojo humano. Es posible agregar, mover, eliminar, cortar, pegar y limpiar fotogramas.
- En cuanto a las *escenas*, son las partes en las que se divide una película Flash. Cada una tiene una línea de tiempo independiente (para mayor explicación sobre líneas de tiempo ver el punto 6.1.3 La línea de Tiempo y las Interpolaciones).

Por lo tanto, dependiendo de la parte de la película en que se esté, y el instante de tiempo seleccionado, los elementos del escenario no serán los mismos, pudiendo haber cambiando su posición, tamaño, o que hayan aparecido o desaparecido objetos.

Normalmente en un documento Flash que no requiera muchos cambios de imágenes, o animaciones diferentes, se emplea una sola escena, y no varias. Esto es debido a que cada escena está en un documento Flash por separado, y al publicar a *.swf, las líneas de tiempo de las escenas se unifican. Esto tiene una serie de desventajas como: el aumento del tamaño del archivo ejecutable, que ralentiza su procesamiento, además de que requiere un control más específico sobre la navegación entre las escenas, ya que puede producirse errores al interpretar el código de ActionScript de una escena sobre otra, por tener la línea de tiempo unida.

*Nota: por defecto, todas las escenas reciben el nombre de "Escena i" donde i es un número secuencial que comienza en 1. Si se desea cambiar el nombre de una escena, se debe ir a **Ventana > Paneles > Escena**, y hacer doble clic en el nombre que se quiera cambiar, para poder editarlo.*

6.1.3 La línea de Tiempo y las Interpolaciones

La línea de tiempo es el elemento que organiza y controla el contenido de un documento a través del de sus componentes principales que son: los fotogramas, las capas y la cabeza lectora. Sin embargo, esta pestaña tiene muchos elementos que se deben conocer, y se muestran en la siguiente figura:

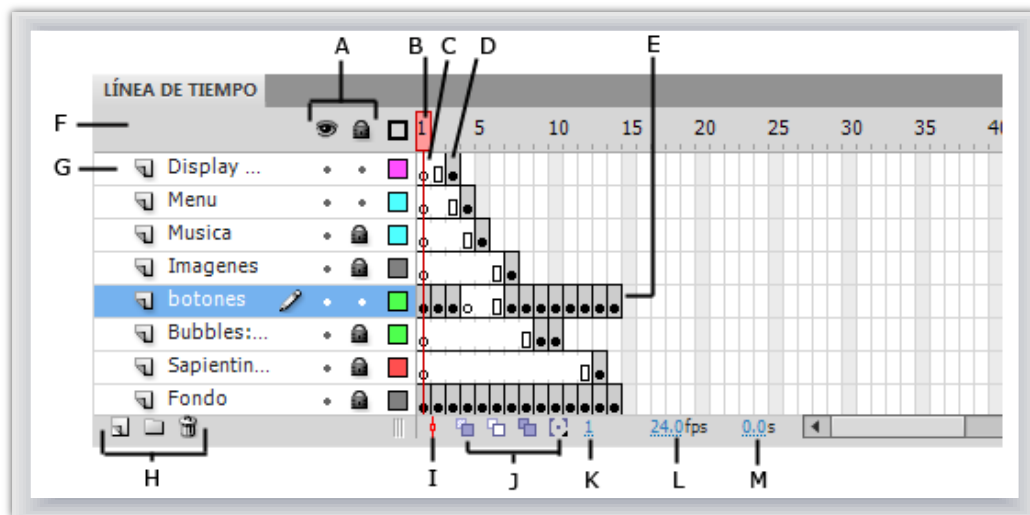


Figura 33. Partes de la línea de tiempo.

- A. *Botones de control.* Permiten mostrar/ocultar y bloquear/desbloquear capas, así como marcar el contenido de las capas con contornos de diferentes colores.
- B. *Cabeza lectora.* Se desplaza por la línea de tiempo, mientras se reproduce el documento, para indicar el fotograma que se muestra en cada momento en el escenario. Para que aparezca un determinado fotograma en el escenario, se debe mover la cabeza lectora hasta ese punto.
- C. *Fotograma clave vacío.*
- D. *Fotograma clave con objetos.*
- E. *Animación fotograma a fotograma.* Conjunto de fotogramas seguidos.
- F. *Encabezado de la línea de tiempo.* Muestra los números de fotograma de la animación.
- G. *Capas de la escena.*
- H. *Botones de organización.* Para añadir o borrar capas, y carpetas para las capas.
- I. *Botón para desplazarse hasta la cabeza lectora.*
- J. *Botones de papel cebolla.* Para ver varios fotogramas a la vez, de forma translúcida.
- K. *Indicador de fotograma actual.*

- L. *Indicador velocidad de fotogramas*. Afecta a toda la línea de tiempo, de modo que para separar secciones a diversas velocidades se debe de realizar en escenas distintas.

Puede diferir de la real, si el sistema no puede reproducirla de forma adecuada.

- M. *Indicador de tiempo transcurrido hasta el fotograma actual*.

Las capas son como bandas de película apiladas unas sobre otras, y cada una contiene una ilustración independiente de las demás, que aparece en el escenario. Esto quiere decir, que las imágenes de una capa que se encuentre en la lista por encima de otra, aparecerán colocadas superpuestas sobre las imágenes de las capas inferiores. Se pueden agrupar en carpetas.

Sólo puede haber una capa activa a la vez, para modificar sus objetos. Para modificar un elemento de otra capa, se deberá de seleccionar esa capa previamente.

Hay tipos de capas especiales, como son las capas de máscara. Como su nombre indica son capas que incluyen objetos utilizados como máscaras para ocultar partes seleccionadas de capas inferiores asociadas (capas enmascaradas). La parte visible será la que no esté cubierta por la máscara.

Los fotogramas clave, se utilizan cuando se quiere realizar un cambio en un punto determinado de la película para crear nuevas instancias de los objetos de una capa. Por ejemplo, si un objeto debe estar diez fotogramas en una misma posición, y se desea que pase a otra a partir del número once, ahí se debe de incluir un fotograma clave, y luego mover el objeto, para que se fije esa modificación. Si no, el cambio realizado también alterará la posición del objeto en los diez primeros fotogramas. Fotogramas vacíos indican un instante de tiempo, de una capa que no tiene objetos que mostrar.

Un *fotograma clave de propiedad* es un fotograma clave en el que se declaran cambios en los atributos de un objeto para una animación (posición, rotación, sesgo, tamaño, color, transparencia...). Para editar un fotograma clave de propiedad, se puede hacer en el escenario, en el editor de movimiento o en el inspector de propiedades.

Flash puede rellenar automáticamente los valores intermedios de la propiedad entre dos fotogramas clave, para generar animaciones sin cortes. Un ejemplo sería colocar un fotograma clave con un círculo y, dejando un cierto espacio entre medias en la línea de tiempo, colocar otro círculo, con otro fotograma clave, más adelante en el eje x. Esta vez, el círculo se encontrará desplazado hacia la derecha. Al crear la animación, Flash calculará el valor de la posición para los fotogramas intermedios, para que se cree una animación fluida de movimiento. Esto es lo que se denomina, *interpolar*, y el conjunto de fotogramas se denomina *interpolación de movimiento*. También se puede aplicar a las formas de un objeto (modificando el tamaño, o

dibujando otra imagen en el fotograma clave destino), siendo una *interpolación de forma*. Y las denominadas *interpolación clásicas*, son como las de movimiento, pero permiten efectos más complejos, siendo también más difíciles de crear.

Para crear una interpolación, se selecciona el fotograma de la capa donde está el elemento que se desea animar, y en el menú que se despliega al pulsar el botón derecho del ratón, se selecciona el tipo de interpolación. Aparecerá un grupo de interpolación, que irá desde el fotograma clave seleccionado, a un segundo fotograma clave que se debe de colocar previamente más adelante en la línea de tiempo. En ese fotograma del final, se debe de realizar la modificación sobre el objeto (cambiar la posición, el color, la forma...), y una vez hecho esto, se puede ver el resultado de la animación, arrastrando la cabeza lectora por la línea de tiempo, o de forma automática, seleccionando el primer fotograma de la interpolación y pulsando "Intro". Si los fotogramas se distribuyen de forma homogénea, la velocidad de la animación será uniforme.

Si el siguiente fotograma clave, después del que se selecciona como inicio del grupo de interpolación, es uno vacío, la animación quedará incompleta, y no funcionará.


Opciones como la aceleración, o si se debe repetir indefinidamente o no, aparecerán en el inspector de propiedades, cuando la animación esté seleccionada. Otra forma para indicar que no se repita indefinidamente, es incluir en el último fotograma de la animación el código `stop()`.

Los tipos de objetos que se pueden interpolar son instancias de símbolo (los gráficos, clips de película y botones), y los campos de texto. No todos los elementos soportan cualquier interpolación. Hay restricciones como la rotación, que se emplea en clips de película solamente, y el movimiento 3D que requiere que el proyecto Flash esté configurado para su reproducción en Flash Player 10 o posterior, y emplee ActionScript 3.0.


En el caso de interpolaciones de movimiento, si la animación es sobre la posición, si se mueve el objeto con el ratón, aparecerá una línea de trazado, que muestra el movimiento por el escenario de la imagen. Se puede editar el trazado del movimiento, variando la posición de los puntos (anclajes), que simbolizan la posición exacta del centro del objeto en cada fotograma, con la herramienta Subselección.


En general, las interpolaciones facilitan mucho el trabajo con animaciones, evitando tener que colocar de forma manual el valor de todos los fotogramas, para dar esa sensación de fluidez.


Al observar la línea de tiempo se pueden identificar las animaciones de un documento, a partir de los colores y los símbolos de los fotogramas. A continuación se hace un breve resumen de los tipos más frecuentes de interpolación:


- *Interpolación de movimiento.* El punto negro indica que tiene asignado un objeto destino, mientras que los diamantes son para fotogramas clave de propiedad, o para indicar el último de la interpolación. 

Un punto blanco al principio, implica que el objeto destino se ha borrado.

- *Interpolación clásica.* 

Si la línea es discontinua, es que está incompleta o se ha interrumpido, normalmente porque falta el fotograma clave final. 

- *Interpolación de forma.* 

- Un fotograma está representado por un punto negro. Los fotogramas gris claro que siguen a uno clave, contienen el mismo contenido, sin cambios. El último del grupo se representa con un rectángulo vacío. 

A la hora de crear una animación, utilizando varios símbolos, cada uno de ellos debe estar situado en una capa distinta. Es aconsejable dejar una capa con las imágenes estáticas, y recurrir a capas adicionales para contener los objetos animados independientes.

En caso de que se añada animación a un elemento de una figura, Flash añadirá capas superiores e inferiores, separando los objetos estáticos, respetando el orden de apilamiento inicial.

Para manipular los fotogramas en la línea de tiempo, se pueden hacer varias cosas:

- *Añadir fotogramas:* Basta con pulsar el botón derecho del ratón, cuando el puntero se encuentra sobre la posición en la que se quiere añadir el fotograma. Se podrá seleccionar entre incluir uno vacío o clave, y también aparecerá la opción de borrado de fotogramas, o de vaciar un fotograma clave, entre otras cosas.
- *Mover a otra posición o capa:* Se selecciona el recuadro correspondiente, y manteniendo pulsado el botón izquierdo del ratón, arrastrarlo a la posición nueva.
- *Copiar/pegar un fotograma, o un conjunto de ellos:* Se puede o bien seleccionar y pulsando el botón derecho del ratón, elegir “copiar fotogramas”, o bien pulsar *Alt*, o la tecla función de un Macintosh y arrastrar donde se quiere pegar.
- *Cambiar longitud de una tira de fotogramas:* Se selecciona el fotograma a partir del cual se quiere modificar la longitud, se pulsa la tecla *Ctrl*, y se arrastra el ratón, hasta la posición deseada. Si hay más fotogramas clave a la derecha de la selección, estos supondrán el límite máximo que se puede ampliar. No se sobrescriben otros fotogramas, por lo que esta variación se deberá hacer aprovechando fotogramas vacíos, o acortando, dentro de la selección.

Los clip de película que se crean en Flash, tienen su propia línea de tiempo. Esta línea, queda anidada como elemento secundario, de la línea de tiempo principal del documento. Lo mismo ocurre si un clip está compuesto por otros símbolos de clip de película.

La relación entre los clips es jerárquica. Si se altera el principal, esto afectará al clip secundario. Esto quiere decir que si por ejemplo, en una animación de un coche, el nivel interno habrá animaciones que serán ruedas rotando sobre sí mismas. Y en un nivel superior, estará el coche completo, desplazándose por el eje x. El desplazamiento se hace sobre el clip “contenedor” (el coche), y hará que las ruedas se desplacen, además de la rotación interna que tienen.

Nota: Se puede ver la estructura anidada de clips, si se selecciona “Mostrar definiciones de símbolo” en el menú del panel.

6.1.4 La Biblioteca y sus Símbolos

El panel denominado biblioteca (**Ventana > Biblioteca**), es donde se guardan y organizan los símbolos creados en Flash, instancias, además de las imágenes importadas al proyecto. Sirve para mantener una copia de todos estos elementos, y que se puedan volver a utilizar en el documento activo, o en otro, ya que se pueden importar al proyecto Flash actual, los elementos de la biblioteca de otro archivo *.fla (**Archivo > Importar > Abrir biblioteca externa**). También se pueden compartir bibliotecas a través de URL.

Incluye un buscador por nombre (los nombres de los elementos de la biblioteca deben ser únicos), y permite la organización en carpetas por parte del usuario. Asimismo se puede consultar la frecuencia con la que se utiliza un elemento determinado, y se permite la ordenación por nombre, tipo, fecha, número de usos, e identificador de vinculación de ActionScript.

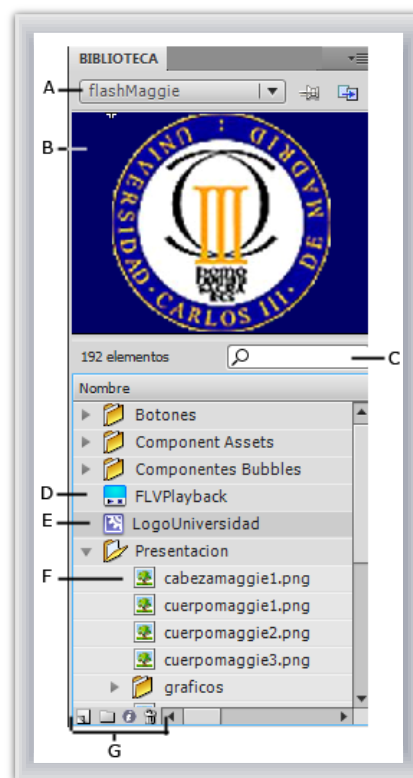





Figura 34. Panel de biblioteca.

En la figura anterior, se muestra el panel de la biblioteca. Está formado por:

- A. *Menú desplegable*. Muestra el nombre del archivo *.fla, al que está asociado la biblioteca que se muestra. Si hay varios abiertos, se podría mirar en la biblioteca de todos ellos, sin cambiar el documento activo.
- B. *Zona de visualización*. En esta parte se ve una vista previa del símbolo seleccionado.

- C. *Campo de búsqueda.*
- D. *Componente de vídeo.* Sirve como reproductor básico de Flash para películas.
- E. *Símbolo de clip de video.*
- F. Imagen *.png importada a la biblioteca.
- G. *Botones de control.* Para añadir carpetas, borrar elementos seleccionados de la biblioteca...

Un símbolo se refiere a gráficos, botones o clips que se crean en Flash, y pasan a formar parte de la biblioteca. Todos ellos tienen una línea de tiempo propia, con sus respectivas capas. Los tipos que hay son:

- *Gráficos* : Se usa para imágenes estáticas ya que no incluyen línea de tiempo, por lo que añaden poco tamaño al archivo *.fla.
- *Botones* : Para crear botones interactivos, con diseños gráficos diferentes para cada estado. Tiene una línea de tiempo especial (más detalle ver el punto 6.1.5 Crear Botones).
- *Clips de película* : Son animaciones, con sus propias líneas de tiempo, que pueden contener otros elementos interactivos. Se puede colocar una instancia dentro de la línea de tiempo de un botón, si se quiere crear uno animado.

Una *instancia*, es una copia de un símbolo que se usa en el escenario, como parte de otro símbolo, o por sí solo. Se crea arrastrando un símbolo desde la biblioteca a la zona de dibujo. En el inspector de propiedades se les puede modificar y asignar un nombre, y es ese nombre el que se utilizará en ActionScript, para hacer referencia al objeto en concreto. Cuando se edita un símbolo, se actualizan todas las instancias que haya sobre él, pero el modificar una instancia, no tiene repercusión sobre el símbolo del que proviene.

Es conveniente pasar las imágenes externas a símbolos, y utilizarlas ya convertidas, debido a que esto reduce el tamaño final del archivo una vez publicado y acelera la reproducción, ya que los símbolos sólo se descargan una vez en el Flash Player. Para convertir un elemento en símbolo se debe pinchar en **Modificar > Convertir en símbolo**, y seleccionar el tipo deseado.

En resumen, el contenido en la biblioteca son símbolos creado por el usuario, importados de otras bibliotecas, o que se han añadido a partir de ficheros de imágenes, que pueden ser reutilizables en cualquier momento y documento.

Además de estos elementos, Flash ofrece componentes prediseñados, para su uso en formularios, y aplicaciones. Estos son clips de película compilados con una funcionalidad definida, como: casillas de verificación, barras de desplazamiento, reproductores de video, menús desplegables, botones, listas, etc. Esta lista se muestra en el panel de componentes, que normalmente no aparece abierto en el programa, por lo que se debe seleccionar **Ventana > panel Componentes**. Para utilizar un elemento, se deberá arrastrar al escenario, o al panel de la biblioteca.

6.1.5 Crear Botones

Los botones son elementos interactivos muy importantes en cualquier interfaz, por lo que se ha de tener especial cuidado con ellos. Son clips de película que constan de cuatro fotogramas, cada uno con una función específica. Los tres primeros: arriba, sobre y presionado, definen el estado del botón cuando el puntero no lo toca, cuando está sobre él, y cuando es pulsado, respectivamente. El último (zona activa), precisa el área de respuesta del botón, que será invisible al publicar el documento Flash.

La siguiente figura muestra el cambio entre los cuatro fotogramas de un botón, que tendrá un color u otro, según si está en reposo, presionado, o si pasa el puntero del ratón sobre él.



Figura 35. Fotogramas para animar un botón.

Para usar un botón, se debe colocar una instancia del mismo en el escenario, y asignarle acciones (esto último se hará mediante código, en respuesta a un *MouseEvent*, lo que se explica en detalle en el punto 6.2.3 Eventos).

Para crear un botón nuevo, se debe hacer **Insertar > Nuevo símbolo**, elegir “botón” y asignar un nombre.

Una vez hecho esto, la línea de tiempo será algo similar a lo que se ve en la parte izquierda de la Figura 35, solo que habrá una sola capa con un único fotograma clave vacío en “Arriba”. Se deberán rellenar los cuatro fotogramas del botón con instancias de símbolos, o dibujando una nueva forma, pero no con otro botón. Se pueden poner fotogramas clave como en la figura de ejemplo, para diferenciar el color del botón en cada caso.

En el cuarto fotograma, lo más sencillo es copiar, por ejemplo, el primer fotograma, y pegarlo aquí. En caso de que los botones sean figuras más complejas, y se desee un contorno ajustado, se recomienda pintar a mano ese contorno, ya que Flash por defecto, interpreta como rectangulares todos los símbolos que se peguen en el fotograma de la zona activa. De todas

formas, esta área, no tiene porque corresponder exactamente a la posición en la que se encuentra el botón, pudiendo estar desplazada, en caso de que interese colocarlo así.

Para hacerlo interactivo, se debe recordar que el símbolo que se va a animar, tiene que ser un clip de película, que puede estar compuesto por otros clips de película, o gráficos, que serán los objetos individuales a animar, y que sólo puede existir un objeto animado en cada capa. A partir de esto, simplemente se debe de añadir la animación al fotograma correspondiente (normalmente en “Sobre”, cuando el ratón pasa por encima, o una vez se ha hecho el clic, en “presionado”), donde se pretende que aparezca, y añadir diversas capas para crear imágenes más complejas.

Como consejo de diseño, conviene que la figura que corresponde a un botón se sitúe en la posición (0,0) del escenario, para poder colocarlo fácilmente, sobre todo, si su posición depende del código ActionScript. Además, para crear un “efecto 3D”, si en el fotograma de “Presionado” se reduce ligeramente el tamaño del la figura que forma el botón, dará la sensación de profundidad cuando se pulse en la interfaz.

Se puede añadir un sonido a un estado del botón, seleccionando el fotograma correspondiente, y eligiendo un sonido de la lista que aparece en **Ventana > Propiedades > Sonido**.

Si se quiere incluir texto en un botón, el texto debe estar dentro de los fotogramas del botón, preferentemente como otra capa, para que cuando pase el ratón por la zona donde están las letras, siga activándose. Esto no ocurre, si se quiere utilizar una instancia de un símbolo de botón genérico en el escenario, y se desean diferenciar, colocando con la herramienta “texto”, un nombre sobre el botón. La zona que ocupe el texto, no será interactiva.

Por defecto, Flash desactiva todos los botones del documento, de forma que al hacer clic no responderá a los eventos del ratón. Esto facilita la selección y manipulación de los botones, por lo que se recomienda mantenerlo así, activándolos cuando se quieran hacer pruebas (se activa en **Control > Habilitar botones simples**), o probándolos en la publicación del proyecto, o reproduciéndolos en la ventana de vista previa de la biblioteca.

6.1.6 Dibujar Figuras

Flash dispone de herramientas de dibujo para crear y modificar imágenes, pero antes de comenzar a dibujar, es necesario comprender la diferencia entre gráficos vectoriales y mapa de bits, y cómo afecta la organización de capas y los modos de dibujo, a la creación de ilustraciones.

Flash trabaja con *gráficos vectoriales*, que describen las imágenes mediante líneas y curvas, con propiedades de color y posición, que se denominan *vectores*. Al editar estas imágenes, no se pierde calidad, debido a que las líneas y curvas que describen su forma se corrigen de forma proporcional.

Los *mapas de bits*, son imágenes compuestas por píxeles, que son puntos de color organizados en una cuadrícula. En su resolución original, el ojo humano no distingue entre los puntos, percibiendo sólo el conjunto. Como los datos que definen la ilustración están fijos en una cuadrícula con un tamaño determinado, al modificar la imagen, la calidad se reduce. Sobre todo al cambiar de tamaño, la cuadrícula se redistribuye. Al ampliar, se duplican los píxeles por zonas, y la figura será una composición de pequeños recuadros apreciables por separado.

En cuanto a los modos de dibujo, hay dos tipos que se pueden seleccionar desde el panel de herramientas:

- *Modo de dibujo combinado* (ver Figura 36): Es el modo que hay por defecto. Las formas gráficas creadas en la misma capa, se fusionan cuando se solapan de un modo destructivo. Esto quiere decir que la forma que queda por encima, corta la parte que cubre del gráfico inferior.

Si tiene borde y relleno, ambos se consideran elementos independientes, que una vez dibujados se pueden mover, dividir y transformar por separado.

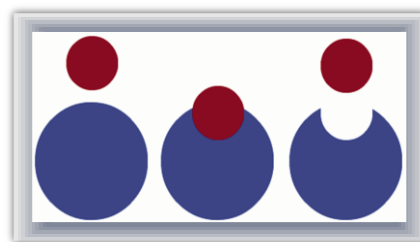



Figura 36. Modo de dibujo combinado.

- *Modo de dibujo de objetos*  (ver Figura 37): Aquí se pueden crear formas denominadas *objetos de dibujo*, que no se combinan de forma automática cuando se superponen, por lo que se pueden manipular por separado. En este caso, el borde y el relleno no son independientes, y Flash rodea con un cuadro delimitador la figura creada con este modo, para diferenciarlo del modo anterior.

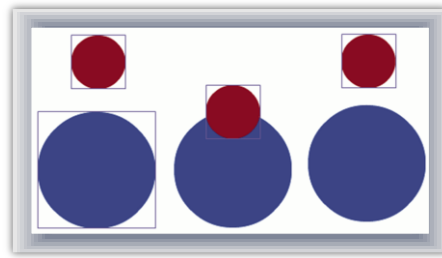





Figura 37. Modo de dibujo de objetos.

Se puede transformar una figura creada con el modo combinado en una forma del modo de dibujo de objetos, si con la imagen seleccionada se pulsa en **Modificar > Combinar objetos > Unión**. Hecho esto, la forma se trata como un objeto de dibujo vectorial que no altera su forma al interactuar con otras figuras. Este comando sirve para unir dos o más gráficos.

En Flash, se pueden dibujar figuras, a partir de formas geométricas simples, con herramientas como: óvalo , rectángulo , línea , etc. Y se puede controlar que, por ejemplo, la inclinación de la línea esté limitada a múltiplos de ángulos de 45°, pulsando la tecla Mayus., a la vez que se arrastra con el ratón para dibujar la línea. O se pueden modificar de forma proporcional el área de una figura arrastrando desde una esquina con el ratón, a la vez que se mantiene pulsada la tecla Mayus. Sin embargo, para realizar figuras realmente complejas de forma manual, se tiene que conocer el uso de los trazados.

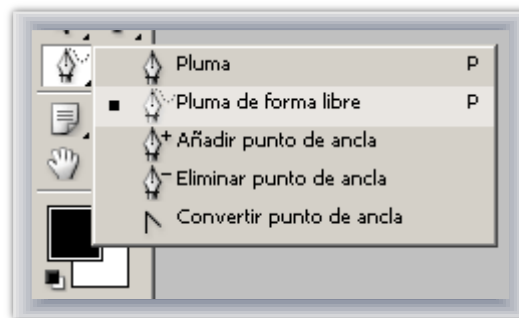


Figura 38. Icono de la herramienta pluma, con la que se hacen los trazados.

Los *trazados* son líneas compuestas de uno o varios segmentos rectos o curvos, cada uno de ellos marcados por puntos de anclaje, que fijan la línea en su lugar. Se pueden realizar formas abiertas (mancando los puntos finales: A), o cerradas muy complejas con este método. Cada punto de ancla (B, ancla seleccionada y C no seleccionada), así como los puntos de dirección (E) que aparecen en el extremo de las líneas de dirección (F), se pueden arrastrar, ya que el ángulo y la longitud de las líneas de dirección son las que definen la forma y el tamaño de los segmentos curvos del trazado (D).

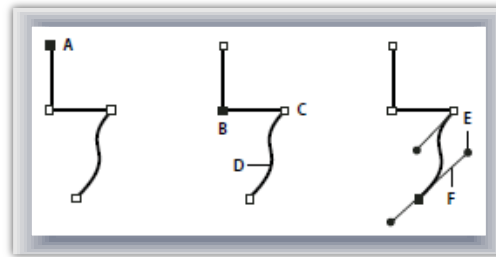


Figura 39. Componentes de un trazado.

Un punto de curva, siempre tiene dos líneas de dirección, que se mueven juntas como una sola unidad recta, ajustándose de forma que se mantenga una curva continua en el punto de anclaje.

Un punto de vértice, dependiendo de si une dos, uno o ningún segmento curvo, puede tener dos, una o ninguna línea de dirección, ya que cada línea se encarga de ajustar la curva que se encuentra en el mismo lado del punto que la línea de dirección.

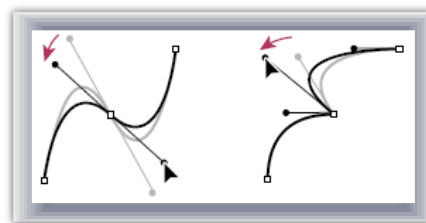


Figura 40. Ejemplo de dibujo con pluma. Punto curvo (izq.), y punto de vértice (der.).

Las líneas de dirección son tangentes a la curva, por lo que su ángulo, determina la inclinación de la curva, mientras que la longitud determina la profundidad de la misma.

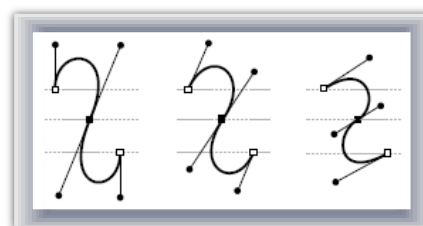





Figura 41. Ejemplos de cambio de inclinación.

Al dibujar con la herramienta pluma, cada clic inicial será un anclaje. Para dibujar líneas rectas, simplemente se deben ir marcando los puntos. Si se desea que la pendiente de las líneas sea múltiplos de 45° , después de haber marcado el primer punto, en los demás se tiene que mantener apretada la tecla Mayus. Cuando se quiere dibujar una curva, al hacer el punto de ancla, se debe arrastrar sin soltar el botón del ratón, para que aparezcan las líneas de dirección, y poder ajustar la forma de la curva, antes de seguir con el siguiente punto del trazado.

Para terminar el dibujo, se puede hacer de varias formas:

- Si se quiere dejar abierto, haciendo doble clic en el último punto y, después, en la herramienta pluma del panel de herramientas. O bien, presionando la tecla Ctrl (Windows) o Comando (Macintosh), y haciendo clic fuera del trazado.
- Si se quiere cerrar, se debe colocar el puntero sobre el punto hueco de ancla, que corresponde al primero que se hizo, y hacer clic cuando aparezca .
- Si se quiere terminar sin haber cerrado la figura, seleccionar **Edición > Anular todas las selecciones**, o bien elija otra herramienta del panel.




Se pueden añadir más puntos de ancla al trazado, o eliminar alguno de los existentes, usando la herramienta correspondiente del panel:  , .


También se pueden convertir puntos de vértice en puntos de curva y viceversa, con el icono ^, del menú pluma desplegable.

Conseguir un buen resultado con este tipo de trazados es una tarea ardua, pero el resultado merece la pena.



Figura 42. Personaje Akio, cortesía de www.nanoda.com.

También se pueden realizar dibujos a mano alzada con la herramienta lápiz  o pincel  aunque estos suelen utilizarse más cuando se dispone de una tabla de dibujo táctil, conectada al ordenador, para poder reconocer la presión del puntero con el que se dibuja, y la anchura del trazo: .

Para editar figuras ya creadas, se puede usar el icono de transformación libre  para escalar, rotar, o sesgar; o remodelarlas si está la herramienta Subselección en uso. Para ello, al pasar el puntero del ratón por el borde de una figura, si aparece una curva, se podrá modificar el lado curvo sobre el que esté el ratón. Si aparece un ángulo, será el caso de vértice del dibujo.

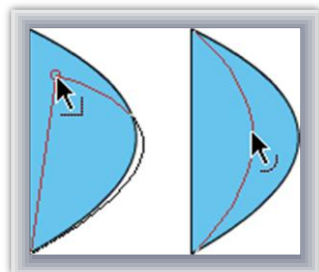




Figura 43. Ejemplo de remodelado de objetos.



En caso de que los trazos del dibujo no hayan quedado muy limpios, y la forma obtenida es más irregular de lo que se quería, se puede recurrir al suavizado de líneas.

Seleccionando la herramienta Selección , aparecerá en la parte inferior del panel de herramientas, el icono suavizar . Cada clic posterior hace que las líneas de la selección se suavicen progresivamente. Se pueden variar los parámetros del suavizado en **Modificar > Forma > Suavizar**.

En caso de querer enderezar alguna curva, utilice el icono .

Un efecto que se puede usar tanto en bordes de figuras, como en el relleno, para difuminar un poco el acabado de la figura, y que no sea tan artificial, es expandiendo o contrayendo las líneas o rellenos. Con la forma seleccionada, ir a **Modificar > Forma > Suavizar bordes de relleno** y fijar la distancia (número de píxeles que ocupará el suavizado), número de pasos (cuanto mayor sea el número, más suave será el efecto), y expandir o hundir (para decidir si la figura se hará más grande o más pequeña al difuminar el borde). Para hacer esto en bordes, primero se tienen que convertir en rellenos en **Modificar > Forma > Convertir Líneas en rellenos**.

Así se consiguen buenos acabados en las figuras, pero amplía el tamaño del archivo.



En cuanto a si se ha cometido algún error de dibujo, se debe de usar la herramienta borrador  o grifo . El grifo, a diferencia del borrador, hace un borrado selectivo según las opciones elegidas. Se puede limitar su uso en bordes, rellenos, las figuras completas, o sólo a los rellenos dentro de un área previamente seleccionada, sin afectar lo demás.


6.1.7 Colores y Degradados

En el punto anterior, se ha explicado en detalle, distintas herramientas de dibujo, transformación, edición, efectos de mejora y borrado, necesarias para crear imágenes complejas en Flash. Pero una vez dibujado el contorno, es necesario conocer el manejo de los colores y rellenos del que dispone el programa, para conseguir buenos acabados.

Cuando se trabaja con los colores, en realidad se están ajustando valores numéricos dentro de la gama de colores del dispositivo que lo produce (unos pueden tener la gama RGB y otros como las impresoras es la CMYK).

El software de Flash CS4 Professional permite manipular colores de las gamas RGB o HSB. Se pueden elegir colores para aplicarlos a los bordes y al relleno de las figuras que se van a crear, o que ya están en el escenario. Para aplicar un color a una forma, se pueden hacer:

- Aplicar un color sólido, un degradado o un mapa de bits al relleno de una figura. En caso del mapa de bits, será necesario haberlo importado previamente a la biblioteca del documento.
- Crear una forma con contorno y sin relleno con la opción , como relleno.
- Crear una forma con relleno y sin borde con la opción , como tipo de contorno.
- Aplicar un relleno de color sólido a un texto.

Con el panel Color, se puede manipular la paleta de colores de un archivo Flash, y modificar el color de bordes y rellenos sólidos y con degradado en los modos RVA (RGB es español) y MSB (matiz, saturación, brillo). Para acceder a ello, se puede seleccionar el icono  del selector de color, en el control *Color de trazo* o *Color de relleno* del panel de herramientas, o del inspector de propiedades, o seleccionar directamente el panel de color:

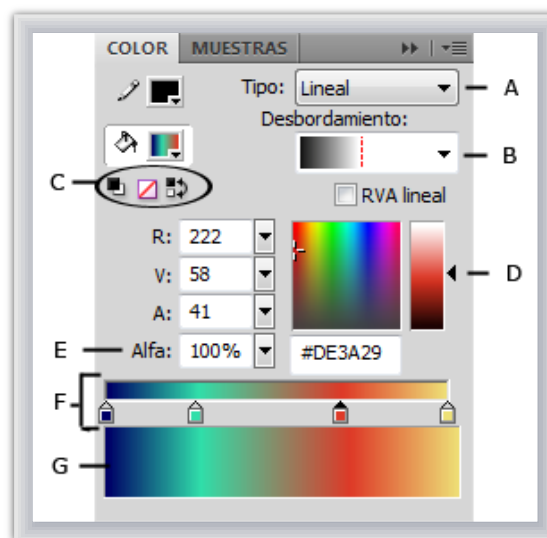






Figura 44. Panel de color con degradado seleccionado.

Los elementos del panel más importantes son:

- A. *Tipo de color.* A elegir entre: ninguno, color sólido, un degradado lineal, uno con forma radial, o un relleno con mapa de bits (repite en forma de mosaico, dentro de la figura, un gráfico importado a Flash).
- B. *Desbordamiento.* Aparece sólo al elegir un color degradado, e indica la forma del mismo, a elegir entre las siguientes tres opciones:  .
- C. *Iconos de opción rápida.*
 - a. Para volver a la configuración predeterminada de relleno blanco y trazo negro  .
 - b. Indica que la figura no tiene relleno  .
 - c. Intercambia los colores fijados de línea y relleno, con cada clic de ratón  .
- D. *Saturación.* Para elegir el tono del color.
- E. *Alfa.* Indica la solidez. Un valor de 100% es un color opaco, mientras que el 0% indica que es totalmente transparente.
- F. *Muestrario de color.* Si está seleccionada una opción de degradado, aquí se podrán añadir indicadores, donde cada uno marque un color determinado. Entre dos marcadores, Flash rellenará las tonalidades necesarias para pasar suavemente de un color a otro.

Se pueden colocar tantos indicadores como se desee, solamente haciendo clic sobre la barra, y para eliminar uno, basta con arrastrarlo fuera del panel. Su posición se puede variar, arrastrándolo hacia delante o hacia atrás por la barra.

Si se quiere modificar su color, se puede hacer en la paleta de arriba, o eligiendo del muestrario que se despliega, al mantener pulsado el botón izquierdo del ratón sobre él.

- G. *Vista previa del color.*

Un color degradado se puede editar ya no sólo en el panel de color, sino directamente en el escenario, cambiando su longitud, concentrándolo en una zona, y moviéndolo de forma que un color del degradado predomine en una zona u otra. Para ello, se debe de seleccionar la herramienta de transformación de degradados, del panel de herramientas y hacer clic en el relleno del objeto gráfico.

Esta opción es posible que no aparezca a primera vista, ya que está en un grupo junto con el icono de transformación libre, como se puede ver en la siguiente figura:

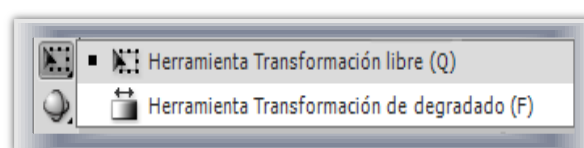


Figura 45. Selección de herramienta de Transformación de degradado.

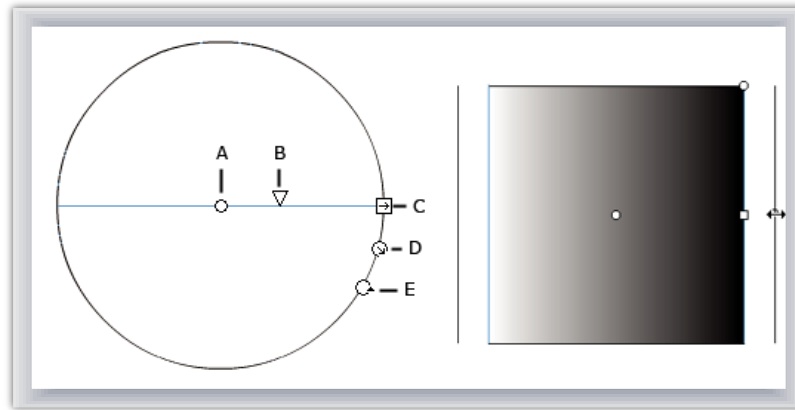


Figura 46. Degradado radial (Izq.) y lineal (Der).


En la figura se muestran el gráfico que aparece en Flash, cuando se quiere modificar un relleno con degradado. Se ha detallado el caso del radial, debido a que tiene un mayor número de componentes, pero los elementos comunes aparecerán en los demás tipos de degradados.

- A. *Punto central.* El icono de desplazamiento del degradado.
- B. *Punto focal.* El selector del punto focal, se muestra únicamente cuando el degradado es de tipo radial. Indica el centro del relleno radial.
- C. *Anchura.* Ajusta la anchura del degradado.
- D. *Tamaño.* Ajusta el tamaño del degradado ampliando o reduciendo el círculo, de forma que el color quedará más concentrado o menos. Sólo aparece en degradados radiales.
- E. *Rotación.* Ajusta la rotación del degradado.

Esta herramienta también es útil en los rellenos de mapa de bits, para aumentar o reducir el tamaño de la imagen, rotarla, sesgar, o cambiar su posición dentro de la zona que ocupe el relleno. Si la figura es más pequeña que esa área, ésta se verá como una composición de mosaico.

6.1.8 Como Asociar Ficheros de ActionScript al Proyecto Flash

La clase principal a la que se asocia el proyecto Flash, será la clase Main. Al reproducir la aplicación, se llama al constructor de la clase y su instancia se añade como elemento secundario del escenario, desde el primer fotograma.

Para indicar el nombre del Main, y su localización, se hace desde el inspector de propiedades del escenario. En el campo *Clase*, se debe escribir el nombre de la clase principal (no tiene porque llamarse “Main”, aunque como norma de diseño, suele mantenerse ese nombre). Esta clase debe de estar en el mismo directorio que el archivo *.fla del proyecto Flash. Pulsando en  se abrirá el archivo en Flash para su edición.

En el caso de algún símbolo que se haya creado en Flash, pero se vaya a controlar desde el código cuando aparece, cuando se elimina, moverlo, añadir algún efecto específico de animación o, en general, cuando la respuesta que deba tener frente a una acción del usuario requiera de cierta complejidad que se tenga que indicar mediante ActionScript, se asocia a una clase *.as. Una vez un símbolo está asociado a una clase, se deberá instanciar desde código, y agregar al contenedor principal de forma explícita. No podrá incluirse una instancia suya en el escenario, a menos que su constructor de la clase sea vacío, ya que si se coloca en el escenario, el compilador considera que se ha hecho una llamada al constructor sin pasar ningún parámetro.

Para poder asociar un símbolo con una clase de ActionScript, se debe seleccionar desde el panel Biblioteca. En el menú que aparece al pulsar el botón derecho del ratón, se escoge “Propiedades”, y esto hará aparecer una ventana como en la siguiente figura:

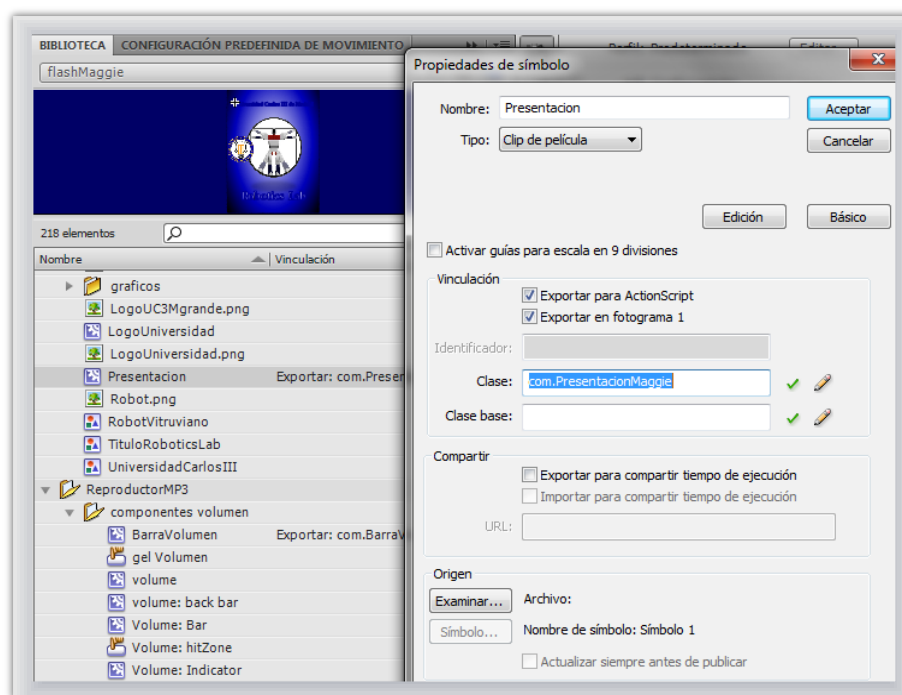


Figura 47. Cómo asociar un archivo *.as a un símbolo Flash.

En esta ventana, se selecciona el cuadro de “Exportar para ActionScript”, para que se active el campo donde indicar la clase. Ahí se debe de escribir la ruta relativa de destino de la clase que se quiere asociar. Para ello, hay que tener en cuenta que el punto de inicio es el directorio donde se encuentre el archivo *.fla del proyecto. Si está en el mismo nivel, bastará con poner el nombre, sin la extensión. Si no, se deberá especificar la ruta de carpetas hasta llegar al archivo, utilizando “.” como separador. En el caso de la figura, estará en el paquete “com”, y nos referimos a la clase “PresentacionMaggie”. También se incluye la opción de “Exportar en el fotograma 1”, para que en caso de tratarse de una animación, la interactividad que ofrezca el código esté disponible desde el principio de la reproducción de ese elemento.

Una vez se introduce, y se acepta el cambio en la ventana, en el panel de la biblioteca aparecerá a la derecha del nombre del símbolo, la ruta de la clase de ActionScript exportada.

6.1.9 Creación de Proyectos y su Publicación

En los puntos anteriores de este manual introductorio a Flash CS4 Professional, se ha explicado los conceptos básicos y el funcionamiento de algunas de las herramientas que se han considerado más importantes de las que dispone Flash, además de unos cuantos consejos de utilización obtenidos durante la práctica.

Por último, se pasará a dar un rápido recorrido a la creación y publicación de los proyectos en Flash.

Para crear un nuevo proyecto, se recomienda fijar la versión de ActionScript a 3.0. De esta forma el reproductor se fijará en la versión de Flash Player 10, por lo que un usuario tendrá que tener instalada esta versión o una superior en su ordenador, para poder visualizar el proyecto (descargable en la web oficial de Adobe¹²).

Para compilar los archivos *.fla, y probar el funcionamiento y la navegación, de todo el proyecto Flash (todas las escenas), se puede pulsar “Ctrl + Intro”, o ir a **Control > Probar película**. Para probar una sola escena, las teclas rápidas son “Ctrl + Alt + Intro”, o ir a **Control > Probar escena**. En caso de usar un Macintosh, pulsar la tecla Comandos en vez de Ctrl.

Esta es una forma rápida de probar el proyecto, sin embargo, los resultados en algunos casos pueden llevar a error, si se prueba sólo de esta forma. Es necesario ver la salida del proyecto una vez publicado, para poder probar plenamente la aplicación y estar seguro de la salida que se obtendrá. Esto se debe a que la publicación es una compilación más exhaustiva, y que opciones como ampliar a pantalla completa, no funcionan con la extensión *.swf, por seguridad.

Para publicar ir a **Archivo > Configuración de la publicación** (ver Figura 48), y elegir el tipo de documento en el que se convertirá, y el nombre y la carpeta de destino y después pulsar el botón *Publicar*.

Una vez que se ha configurado por primera vez la publicación, las siguientes ocasiones se podrá abreviar pulsando **Archivo > Publicar**, sin tener que especificar otra vez la configuración, ya que ésta se guarda junto con el proyecto.

¹² Descarga gratuita de Flash Player - <http://www.adobe.com/es/products/flashplayer/>

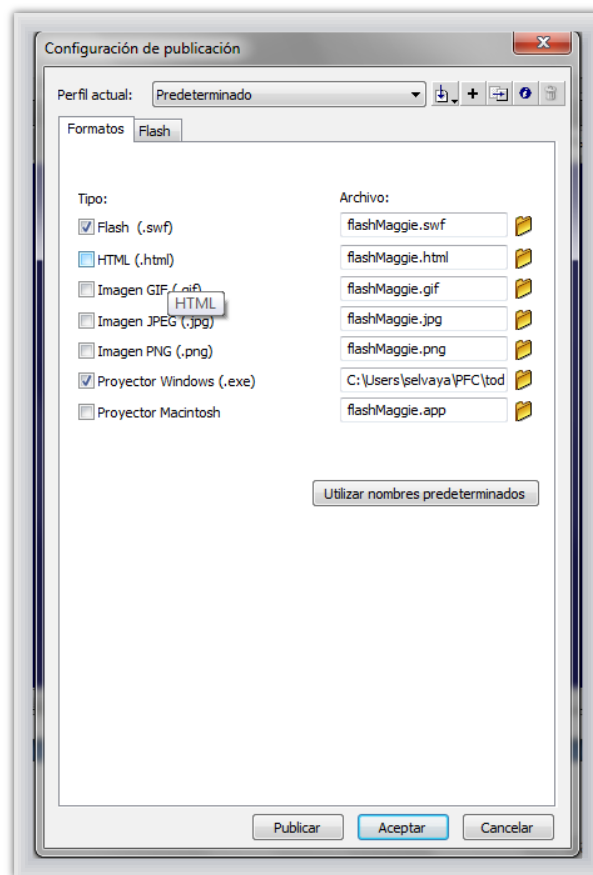


Figura 48. Ventana de Configuración de Publicación.

En caso de que al compilar los archivos Flash, alguno de los archivos que componen el proyecto, tenga algún tipo de error (error en la ejecución del código AS, o intentar acceder a una imagen que no existe, etc.), el panel Proyecto detiene la compilación y se mostrará el error en el panel Salida.

6.2 Breve Manual Introductorio a ActionScript 3.0

Este manual, está orientado para todos aquellos que no hayan programado nunca en este lenguaje, pero si conozcan la programación OO, por lo que se explicarán los puntos más importantes de su versión más avanzada hoy en día, que es la 3.0. Es necesario conocer ActionScript, ya que ofrece los medios para crear interacciones y agregar funcionalidades a Flash, que no serían posibles únicamente manipulando la línea de tiempo.

Los archivos AS son archivos de ActionScript, que se utilizan para mantener el código fuera de los archivos *.fla, lo que reduce el peso del proyecto, además de ser muy útil en la organización del código y en los proyectos donde hay colaboración de un grupo de personas, lo que se espera ocurra en este caso, en el futuro.

6.2.1 Primeros Pasos

ActionScript 3.0 es un modelo de programación orientado a objetos, evolucionado a partir de las versiones anteriores de AS, para dotarle de mayor robustez frente a errores, mayor velocidad de ejecución, y mejor gestión de memoria, entre otras cosas. También tiene un API mucho más completa, con un gran número de ejemplos de uso, que cualquier desarrollador encontrará de gran utilidad. Se puede consultar en la siguiente URL:

<http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/package-summary.html>

Una de los conceptos básicos que se deben de comprender antes de empezar a trabajar con ActionScript, es cómo funciona la jerarquía en niveles de los objetos de visualización en Flash, comenzando con la idea de qué es un contenedor.

Un *contenedor*, es un tipo especial de objeto de visualización, que además de tener su propia representación visual en la pantalla, hace las funciones de recipiente para los distintos objetos gráficos que se le asignen. Si se borra un contenedor, se eliminaría del escenario todas las imágenes que estuvieran “dentro” de él. Si se desplaza, o sufre alguna transformación o animación, esto también afectaría a los elementos incluidos de igual manera.

Pues bien, en Flash, cualquier elemento pertenece a uno u otro contenedor, y estos entre sí asumen una jerarquía en forma de árbol. El contenedor principal es el escenario (objeto *stage*, accesible desde código mediante la propiedad con el mismo nombre que tiene cualquier instancia de *DisplayObject*), y todas las instancias de símbolos colocados en el escenario estarán incluidas dentro de él.

Si trasladamos esta idea al código de ActionScript, el escenario será un objeto similar a un *Sprite* (ver punto 6.2.4 Las Clases Movieclip y Sprite), pero con mayor eficacia en la

representación y el uso de memoria, y una mejor administración de la profundidad y una instancia de la clase principal se incluirá como elemento secundario. A partir de ahí, cualquier instancia a la que se le haya asignado un nombre en el inspector de propiedades, en un segundo plano, el compilador de Flash creará una variable con el mismo nombre que la instancia, accesible igual que si fuera un atributo de la clase principal del proyecto. De esta forma, utilizando ese mismo nombre en el código de las clases, se podrá hacer referencia directa sobre el elemento del escenario que se necesite en cada momento y manipularlo cómo se precise. Esto es básico, para poder realizar un control sobre los eventos, y responder adecuadamente a las acciones del usuario, de forma que el programa Flash esté dotado de interactividad.

De forma análoga, si una instancia de un símbolo, está a su vez compuesta por otros símbolos, estos se consideran en un nivel inferior dentro de la jerarquía, y para hacer referencia a esa instancia, se tendrá que hacer mediante el nombre de la que la contiene. Con un ejemplo, si el símbolo *botonVolumen*, está formado a su vez, por otro símbolo que es *barraVolumen*, para que se pudiera acceder y manipular *barraVolumen* en el Main, se tendría que hacer con la ruta:

```
botonVolumen.barraVolumen
```

En caso de querer acceder desde una clase que no es la principal, tendríamos que obtener una referencia del mismo, y guardarlo en un atributo, si se va a acceder a él más de una vez. Si ese atributo se llama *root*, quedaría de la siguiente forma:

```
root.botonVolumen.barraVolumen
```

Esta sería una ruta completa, pero también se podría hacer referencia de forma relativa en caso de que se estuviera en otra ubicación, dentro de *botonVolumen*, que bastaría con poner el nombre de la instancia de *barraVolumen*.

Desde un punto de vista gráfico, la idea sigue una estructura como la siguiente:

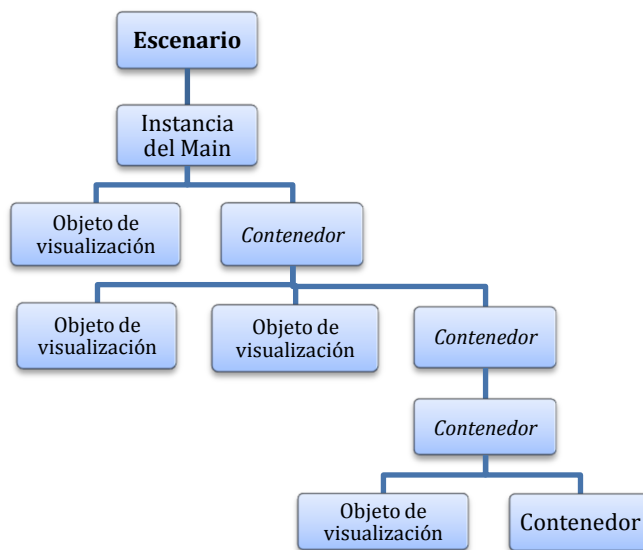


Figura 49. Jerarquía de la lista de visualización.

La línea de tiempo principal de Flash se considera en el nivel cero del reproductor Flash. Las imágenes, gráficos, videos, y cualquier otra cosa que se añada desde el código al escenario, en niveles inferiores, se verán por encima de los objetos del nivel anterior. Para que un elemento gráfico creado en el código (por ejemplo: *dado*), aparezca en el escenario, ese objeto, se ha de añadir al contenedor principal con:

```
contenedor.addChild (dado);
```

Y para borrarlo del escenario, junto con todos sus elementos, sería:

```
contenedor.removeChild (dado);
```

Contenedor deberá ser una instancia de la clase principal (Main). Si no lo es, entonces deberá de ser un elemento intermedio, que haya sido previamente añadido de la misma forma al Main. Se puede formar una cadena de contenedores y elementos gráficos, pero por muchos elementos que tengan, deberán estar conectados en algún punto al contenedor principal, para que se aparezcan en el escenario.

Además de esto, a la hora de colocar varios elementos en el escenario, se debe de tener en cuenta otra cosa. Cuando se añade un objeto gráfico al escenario, si pone otro en ese mismo nivel, al haber sido añadido después, aparecerá por encima del primero, como si se tratara de dibujos en papel transparente. Y si están en niveles diferentes (seguir un esquema similar a la Figura 49), los elementos de visualización de niveles inferiores, se verán por encima de los objetos gráficos de niveles superiores, debido a la sencilla idea, de que los elementos apilados más tarde, se colocan por encima de los demás.

No es necesario especificar el nivel en el código, ya que esto se hace automáticamente, sumando uno, al nivel del contenedor donde se añade el elemento. Sin embargo, se puede hacer de forma explícita utilizando:

```
contenedor.addChildAt (dado, 3);
```

Y para borrar:

```
contendor.removeChildAt (dado, 3);
```

Una vez comprendido el concepto de los niveles, el siguiente punto más importante, es el cambio de fotogramas o de escenas, desde el código. En principio, el código AS se carga en el fotograma cero de la línea de tiempo, y si hay una animación, esta se reproducirá según su línea de tiempo secundaria, pero para cambiar de fotograma en la línea principal de forma explícita, se utilizan las siguientes sentencias:

```
gotoAndStop (numeroFotogramaDestino);
```

```
gotoAndStop ("NombreEscena", numeroFotograma);
```

La primera indica el fotograma al que tiene que saltar Flash en la línea de tiempo, sin que se continúe la reproducción más allá. La segunda es para cambiar de una escena a otra, especificando de igual manera, el instante de tiempo donde debe ir.

Es mejor utilizar estas sentencias, y no las de *gotoAndPlay*, en la ejecución principal, ya que estas hacen que el control de la animación sea más difícil en aplicaciones como la interfaz creada en este proyecto. En caso de querer reproducir clips de película, si se recomienda su uso.

En cuanto a las propiedades básicas de cualquier elemento con componente gráfico en ActionScript, como: *MovieClip*, *Sprite*, *Graphics*, *Shape*, los componentes de Flash, etc. Todos ellos acceden a esas propiedades de la misma forma, a partir de una instancia y con el operador “.”, seguido del nombre de la propiedad, ya sea para asignar un nuevo valor, o para “leer” el valor almacenado:

```
clip.width = 50; //asigna un valor de 50 píxeles de anchura al "clip"
trace(clip.height); //muestra por pantalla la altura de "clip"
```

La diferencia con versiones anteriores radica en que los nombres han cambiado, eliminando el carácter “_” con el que comenzaban todos ellos, de forma que en resumen, son:

- Propiedades de posición: *x*, *y*.
- Propiedades de tamaño: *width*, *height*.
- Escalado y rotación: *scaleX*, *scaleY*, *rotation*.
- Color: *alpha* (transparencia), *transform* (accede a la clase de control del color).
- Control visibilidad: *visible* (*true* o *false*).

Finalmente, unos consejos para evitar problemas en el control de instancias de objetos colocados en el escenario en Flash, desde ActionScript:

- Utilizar nombres exclusivos. El mismo símbolo, en fotogramas diferentes, con el mismo nombre de instancia en cada uno de ellos, está permitido, pero se ha de vigilar su funcionamiento de cerca. Esto es así porque al inicio, la clase principal se carga en el fotograma cero de la línea de tiempo, y al pasar a otro fotograma con la sentencia *gotoAndStop(f)*, ese código deja de estar disponible, siguiendo la ejecución a partir de la siguiente línea de código de la que ha provocado el salto.
- Por el motivo anterior, al trabajar con una interfaz se recomienda que cada sección esté en una clase diferente. Así, al cambiar de fotograma en la navegación, al pasar de una sección a otra, se crea un objeto de la clase, y dentro de esa clase se realiza todo el control de los elementos que correspondan a ese fotograma.
- De la misma forma, al volver a un fotograma donde hubiera un menú, hay que asegurarse de añadir las instrucciones de control de eventos, o no habrá reacción cuando se intente interactuar con algún elemento que antes sí funcionaba, ya que al cambiar de fotograma, el código cargado anteriormente, se pierde.

6.2.2 La Orientación a Objetos

Una vez explicados unas notas generales sobre el lenguaje, se pasará a explicar brevemente su sintaxis y estructura.

En la programación OO, el código del programa se agrupa en clases, que no son más que contenedores. Las instrucciones se dividen según su funcionalidad, de modo que los tipos de funcionalidad relacionados estarán en la misma clase. Las características principales del contenedor estarán almacenadas en variables que serán los atributos, y los métodos o funciones serán las distintas acciones que pueda realizar ese objeto. Para acceder a cualquiera de ellos, se hará mediante el operador “.”.

Para acceder a la posición *x*, del objeto *Coordenada*, denominado *coord*, sería:

```
coord.x;
```

Para acceder al método *sumar (num1, num2)*, de la clase *Matemáticas*, sería mediante la instancia *mat* de esa clase, de la siguiente manera:

```
mat.sumar (2,3);
```

En cuanto a las *variables*, estas se componen de tres partes: el nombre, el tipo de datos, y el valor real almacenado en la memoria del equipo.

Para crear, o declarar una variable en Flash, se usa la siguiente sentencia:

```
var value1:Number = 25;
```

Utilizando la palabra reservada *var*, que indica que se está declarando una nueva variable, de nombre *value1*, de tipo *Number*, y con un valor inicial de 25.

ActionScript es sensible a mayúsculas y minúsculas, por lo que a la hora de declarar variables, clases o funciones, se deberá tener esto en cuenta, para evitar errores de compilación.

En cuanto a los tipos básicos de variables (almacenan un solo dato o campo), y sus valores predeterminados (lo que contienen cuando aún no se ha inicializado), son:

- **String**: Sirve para valores de texto de uno o más caracteres. Valor inicial: *null*. Al considerarse tipo básico, se pueden comparar dos String utilizando directamente signos de comparación *==*, *!=*, *>*, *<*, etc.
- **Number**: Para cualquier tipo numérico, decimal o no. Valor inicial: *NaN*.
- **Int**: Para número enteros. Valor inicial: 0.
- **Uint**: Enteros estrictamente positivos. Valor inicial: 0.
- **Boolean**: Admite los valores *true* o *false*. Valor inicial: *false*.

El resto de clases del API o las creadas por el usuario, su valor sin inicializar será *null*.

Además de estos, se pueden crear variables de otros tipos de datos que almacenan más de un campo, y que normalmente, corresponden a clases creadas en el API de Flash, o bien, a clases creadas por el usuario, de forma idéntica a como se haría en C++. Algunas de estas clases son:

- **Array:** Para colecciones de objetos, sin especificar su tipo. Para crear un array:

```
var miArray:Array = [1, 2, 3]; //valor literal
var someArray:Array = new Array(1,2,3); //usando el constructor
```
- **MovieClip:** Símbolo de clip de película.
- **TextField:** Campo de texto. Se usa para recoger la entrada de texto, para campos cuyo contenido se actualiza desde el código, o como campo estático.
- **SimpleButton:** Símbolo de botón.
- **Date:** Información sobre un solo momento temporal (una fecha y hora).

Los operadores matemáticos, los símbolos para comparaciones, los comentarios, el separar las estructuras con "{ }" y terminar cada instrucción con ";", son reglas igual que las utilizadas en otros lenguajes de programación OO tipo C++ o Java. En cuanto al control de flujo, y la estructura de clases, estas se parecen más a Java, por lo que a continuación se hará un breve repaso de sintaxis, explicando sus diferencias o similitudes con Java, considerando que el lector tendrá unos ciertos conocimientos básicos sobre este lenguaje de programación:

- **Bucles:** Estructura idéntica para bucles *for*, *for each...in*, *while* y *do...while*, con la única diferencia de que la variable contador debe definirse utilizando la palabra reservada *var*, tal como se ha explicado anteriormente. Con *for...in*, cambia un poco, construyéndose de la siguiente forma:

```
var miLista:Array = ["uno", "dos", "tres"];
for (var i:String in miLista){
    trace (i + ": " + miLista[i]);
}
```

Puede recorrer propiedades de objetos o matrices, pero no de clases definidas por el usuario.

- **Sentencias condicionales:** Mismo formato de bloques *if*, *else*, y *else if*, y de sentencias *switch*.
- **Funciones:** Hay dos tipos de funciones en ActionScript 3.0, que dependen del contexto en el que se definen:
 - **Métodos:** Funciones que se definen como parte de una clase o se asocian a una instancia de un objeto.

```
function nombreFuncion(nombreParametro:tipo):valorRetorno{
    //instrucciones
}
```

El tipo de retorno puede ser cualquiera. En caso de que no se devuelva nada, el tipo será *void*.

En las versiones anteriores, se utilizaba otro tipo de función, denominado *expresiones de funciones*. Este código se colocaba directamente en los fotogramas de la línea de tiempo de la instancia sobre la que se quería controlar alguna acción. Ejemplo;

```
var mostrar:Function = function (parametro:String)
    trace(parametro);//función de salida estándar
};
mostrar("hola"); // se muestra "hola", con un salto de línea
```

Ahora en la 3.0, se podría seguir haciendo de esta forma, reuniendo el código en archivos *.as sin formato de clase de forma dinámica similar a Python. Sin embargo, no se recomienda este uso ya que ahora, por ejemplo, no está permitido incluir código en la línea de tiempo de un botón, por lo que es mejor dejar todo el código fuera de la línea de tiempo, y que no esté dividido, para una mejor organización, utilizando una estructura de clases.

- **Cierres de función:** Son funciones que se ejecutan fuera de un objeto de una clase. Contiene todas las funciones, variables, y propiedades necesarias para su cadena de ejecución. Un ejemplo:

```
function foo():Function{
    var x:int = 40;
    function area(y:int):int{ // definición de cierre
        return x * y
    }
    return area;
}
function bar():void{
    var x:int = 2;
    var y:int = 4;
    var producto:Function = foo();
    trace(producto(4)); // llamada a cierre de función
}
bar(); // 160
```

- **Try-catch:** Se usa en la gestión de errores, para capturar excepciones que se pueden producir en el código, como al fallar en un intento de conexión de un socket, para realizar un seguimiento del error, y aplicar acciones determinadas ante el problema, como avisar al usuario de que no se pudo conectar, evitando que el programa acabe de repente. El formato de estas sentencias es:

```

try{
    texto = myByteArray.readBoolean();
}catch (error:EOFError){
    trace (error.toString());
}

```

En cuanto a la estructura general de las clases, estas pueden ir organizadas en paquetes (carpetas), y se debe especificar en cada momento a que paquete pertenece y, en caso de hacer referencia a las clases incorporadas al API de Flash, se deben de importar de forma explícita, mediante la sentencia *import*:

```

import flash.display.*; //se importan todas las clases de display
import flash.display.MovieClip; //se importa sólo MovieClip

```

Esto se debe a que estas clases se importan automáticamente en los scripts adjuntos a la línea de tiempo, pero no a las clases implementadas por separado. Estas clases están en los paquetes *flash.** y los componentes de edición en *fl.**.

Si se pretende usar una instancia de una clase creada del usuario, localizada en otro paquete, también se tiene que importar la clase o el paquete específico, lo cual es una variación respecto a la versión anterior de ActionScript, donde la importación era opcional.

Una vez explicado esto, una clase tendría el siguiente aspecto:

```

package nombrePaquete
{
    public class nombreClase
    {
        //lista de atributos
        public var nombreAtributo:tipo;
        ...
        //constructor de la clase. Puede contener o no parámetros
        public function nombreClase()
        {...}

        //lista de métodos
        public function nombre (param1:tipo, param2:tipo):retorno
        {...}
    }
}

```

Los atributos se pueden inicializar directamente en la instrucción donde se declaran, o dentro del constructor.

El constructor es un método especial, cuyo nombre debe coincidir exactamente con el de la clase, que se ejecuta cuando se crea una instancia de la misma con la palabra clave *new*. Si no se incluye un método constructor, el compilador añadirá automáticamente un constructor vacío (sin parámetros ni sentencias), en la clase.

En cuanto a la accesibilidad de las clases, los especificadores de control de acceso son:

- *public*: Sirve para asegurarse que la accesibilidad será total, desde cualquier clase y paquete.
- *private*: Sirve para restringir el acceso únicamente a las llamadas que provengan del código de la misma clase.
- *protected*: Permite el acceso de la clase actual y derivadas.
- *internal*: Si no se especifica ninguno de los valores anteriores en el código, este es el predeterminado. Significa que sólo estarán visibles para las llamadas cuyo origen sea ese mismo paquete.

La herencia también está contemplada en ActionScript, mediante el uso de la palabra reservada *extends*.

```
public class claseHija extends clasePadre{...}
```

El atributo *override* sirve para redefinir un método heredado y el atributo *final*, para evitar que las subclases sustituyan un método.

También se admite el uso de polimorfismo igual que en Java, y el uso de *super*, para hacer referencia a los métodos de la clase padre. En cambio, el uso de la palabra reservada *this*, indica una referencia a la instancia de la clase donde se ha hecho la llamada a *this*.

Por último, para compilar las clases en ActionScript, se debe probar la aplicación, como ya se ha explicado en el punto 6.1.9 Creación de Proyectos y su Publicación. Si no aparecen errores de ejecución en el panel salida, entonces es que el código se compiló correctamente.

Como se puede observar hasta ahora, aquellos con experiencia en Java o C++, encontrarán ActionScript con una sintaxis muy parecida en sus estructuras elementales. Aunque aquí sólo se ha presentado algunas notas para un conocimiento básico, ActionScript tiene mucho más: definiciones de interfaces, métodos estáticos, creación de clases dinámicas... Pero para la comprensión de la aplicación de este proyecto, no era necesaria una explicación tan profunda.

Sin embargo, en los siguientes apartados se ilustrará sobre el uso de partes algo más específicas, como los eventos, y una introducción al uso de algunas de las clases del API que se utilizan con mayor frecuencia al trabajar con Flash, y que se han aplicado a este proyecto.

6.2.3 Eventos

Un programa de ActionScript, se diseña para dotar de interactividad a una aplicación Flash. Para ello, debe de existir un sistema de control, que permita detectar acciones, datos introducidos por el usuario, o en general sucesos de interés, y ejecutar una serie de instrucciones en respuesta. Los eventos son un mecanismo que permite controlar esto. Es un componente fundamental en cualquier programa Flash, y en este punto se explicará su funcionamiento.

En ActionScript 3.0, los eventos son instancias de la clase *Event*, o de alguna de sus subclases: *MouseEvent*, *ProgressEvent*, *SliderEvent*, *ErrorEvent*...

Los objetos de este tipo no sólo almacenan información sobre un evento determinado, sino que contienen métodos para manipular el evento.

Según si se trata de eventos de componentes o no, se tendrá que utilizar la clase *Event* del paquete *fl.** o *flash.**, respectivamente.

Para detectar si se ha producido un evento determinado, como por ejemplo, un clic de ratón, se creará un objeto de evento, especificando la clase (ej.: *MouseEvent*). De esta forma, cuando se pulse el botón del ratón, el reproductor de Flash distribuirá el evento al *objeto destino*, que será quien estaba “pendiente” de este evento, como por ejemplo, un botón, al clic del ratón sobre él.

Si el destino del evento, está en la lista de visualización, el objeto de evento pasa por la jerarquía, hasta alcanzar su destino, lo que se denomina *flujo del evento*.

Una vez que se ha detectado un evento, para responder, se usan *detectores de eventos*, que son funciones o métodos que se ejecutan cuando un objeto destino recibe un evento. La estructura básica de un detector de eventos sería:

```
function nombreFuncionManejadoraEvento(objetoEvento: tipoEvento):void
{
    //acciones realizadas como respuesta al evento
}
```

Cuando un objeto destino se quiere suscribir a un evento en el código, debe especificar el nombre del detector del evento, y el tipo del evento, de la siguiente forma:

```
objeto.addEventListener(tipoEvento.NOMBRE, nombreDetectorEvento);
```

De esta forma, para mostrar un “hola mundo”, cuando se produjera un clic de ratón, sobre un botón, denominado *btn_prueba*, el código completo sería:

```
//primero se suscribe
btn_prueba.addEventListener(MouseEvent.CLICK, manejadorBtn);
//Código del detector de evento
function manejadorBtn(e:MouseEvent):void{
    trace ("hola mundo");
}
```

Esta forma de registrarse a los eventos en ActionScript 3.0 es la principal diferencia frente a versiones anteriores, ya que presenta un sistema genérico para tratar los eventos, que sustituye a todos los anteriores. Antes, había que sobrescribir métodos específicos como *on()*, *onActivate()*, *onChanged()*, y colocarlos como código, dentro de la instancia del símbolo que iba a generar el evento, ya que no existía el *flujo del evento*, mientras que ahora se puede detectar el evento, en cualquier nodo del flujo. Esto resulta útil cuando un símbolo consta de varios elementos. Por ejemplo, en un botón con un campo de texto, en versiones anteriores, se tendría que añadir un detector tanto al botón, como al objeto de texto, para garantizar que se reciban las notificaciones de eventos de clic que se produzcan. Sin embargo, de esta forma, con un único detector de eventos, se pueden controlar los clics que se produzcan tanto en el campo de texto, como en el resto del botón.

Si en un momento dado, el objeto destino quiere dejar de seguir un evento, puede hacerlo incluyendo la siguiente instrucción en el código:

```
objetoDestino.removeEventListener(nombreDetectorEvento);
```

Es aconsejable eliminar todos los detectores, cuando ya no se vayan a usar más, sobre todo aquellos eventos como *onEnterFrame*, que están constantemente lanzándose, ya que ralentizan la ejecución.

En cuanto al tipo de objeto de evento, se puede acceder a su valor en cadena de caracteres, a ellos a través de la propiedad *Event.type*. Es útil conocer el tipo, para distinguir en el código los objetos de distintos eventos. Las subclases de *Event*, también tienen constantes para representar el tipo de evento asociado a esa subclase.

Tomamos de ejemplo *MouseEvent*, dado que es uno de los eventos más utilizados. Dentro de las constantes definidas en esta clase está: *CLICK*, *DOUBLECLICK*, *MOUSEDOWN*, *MOUSEUP*, *MOUSEOVER*, entre otros... si accedemos a la propiedad *type*, de una instancia de *MouseEvent*, devolverá la cadena de caracteres correspondiente al tipo de evento específico que se produjo.

Otra propiedad muy útil sobre los eventos, es *target* o *currentTarget*, ya que guardan una referencia del destino del evento. A veces el destino es obvio, pero cuando se trata de un objeto de la lista de visualización, es necesario tener en cuenta la jerarquía de esta. Además, en casos en los que se siga un comportamiento similar en respuesta a un mismo evento, para distintos elementos, se puede utilizar el mismo detector de eventos, y dentro de él, con esta propiedad, diferenciarlos para aplicar el comportamiento adecuado para cada destino. Esto evita tener que incluir una función por cada objeto, agrupando el código.

Cuando se trata de elementos de visualización colocados en el escenario, se puede usar *target.name*, que devuelve un valor de tipo String, con el nombre de la instancia que se le haya dado en Flash, de forma que identificarlos se convierte en una tarea muy sencilla.

No hay diferencia entre *target* y *currentTarget* si se aplica en símbolos sencillos. Sin embargo, en caso de aplicaciones con mayor decorado y complejidad en los gráficos, con botones compuestos por varios elementos secundarios, puede haber problemas. En esos casos, no es raro si la propiedad *target* señala a los objetos secundarios de los botones en lugar de a los propios botones. Esto se debe a que siempre selecciona el objeto que se encuentra más lejos del objeto *Stage* como destino del evento, por lo que es mejor usar *currentTarget*, con el evento asociado al botón en sí, para evitar conflictos.

En caso de que vaya a aplicar un mismo código, como respuesta a dos eventos diferentes (por ejemplo, *MouseEvent.CLICK* y *Event.COMPLETE*), se puede usar un único detector de evento. Esto es posible, si el tipo de evento que recibe como parámetro es *Event*, la clase padre de la que heredan. Así, se admitirán eventos de todo tipo, sin hacer distinción, y sin causar excepciones en la ejecución, y con la propiedad *type*, se podrá diferenciar con sentencias condicionales, que hacer para cada caso.

6.2.4 Las Clases *Movieclip* y *Sprite*

Dos de las clases que más utilizará un programador en Flash serán:

- **MovieClip:** Representa símbolos de clip de película. Todos los símbolos de este tipo de la biblioteca, al compilar pasan a ser instancias de *MovieClip*, pudiendo acceder a sus propiedades y métodos de clase, en el código. Algunos de esos métodos y propiedades más importantes son:
 - Reproducir y detener clips de película con *play()* y *stop()*.
 - Mover la cabeza lectora a un punto determinado de la línea de tiempo con *nextFrame()* y *prevFrame()*. Con *currentFrame* se puede saber en todo momento el número de fotograma en el que se está y *numFrames*, devolverá el número total de fotogramas que dura la animación actual.
 - Moverse entre escenas con *gotoAndPlay* y *gotoAndStop* (esto ya se explicó en el punto 6.2.1 Primeros Pasos).
 - Acceder a las propiedades de color, posición, tamaño, etc.

Tiene su propia línea de tiempo y puede utilizar interpolaciones para crear animaciones.

- **Sprite:** Clase nueva en la versión 3.0 de ActionScript, que representa un contenedor básico ligero, similar a la clase *MovieClip*, pero diseñado de forma específica como clase base para los contenedores de objetos.

En este caso, sólo se pueden crear animaciones mediante código.

En versiones anteriores, la clase *MovieClip* era la base de todas las instancias del escenario, pero ActionScript 3.0, si no se necesita una línea de tiempo para el funcionamiento del objeto de visualización, se recomienda utilizar clases como *Shape* o *Sprite*, para mejorar el rendimiento de la representación.

Para utilizarlas en el código, se debe de importar la clase con la instrucción:

```
import flash.display.MovieClip;
import flash.display.Sprite;
```

Y en caso de crear una clase que se quiera beneficiar de las propiedades y métodos predefinidos de *MovieClip* o *Sprite* (es decir, cualquier elemento gráfico que se cree en ActionScript y se añada al escenario), deberá heredar de ellas.

En cualquier caso, se recuerda que ningún elemento de tipo *MovieClip* o *Sprite* creado en el código, estará visible en la pantalla hasta que no se añada a la lista de visualización llamando al método *addChild()* o *addChildAt()*.

6.2.5 Crear Formas Gráficas desde Código. Uso de la Clase Graphics

ActionScript 3.0 permite la creación de gráficos vectoriales (líneas, curvas, círculos, elipses, rectángulos normales y con puntas redondeadas, formas con o sin relleno e incluso líneas y curvas cuadráticas de Bézier) mediante código. La clase *flash.display.Graphics* es la encargada de realizar esta función, y puede realizar figuras muy complejas, pero en este manual sólo se va a explicar el aspecto básico con algún ejemplo, para darlo a conocer y familiarizar al lector con su uso.

Para dibujar se debe de hacer con las clases *Shape*, *Sprite* o *MovieClip* asignando una instancia creada del dibujo, a la propiedad *graphics* definida en cada una de esas clases. Una instancia de *Shape* es la mejor opción para utilizar como lienzo de dibujo, en caso de que no se necesite un objeto de visualización que pueda contener más objetos, ya que no tiene la sobrecarga de la funcionalidad adicional de las otras clases. En ese caso sería mejor usar un objeto *Sprite*.

Los pasos básicos antes de comenzar el dibujo son:

- **Estilo del trazo:** Antes de comenzar a dibujar se debe definir el tamaño y el color de la línea. El método *lineStyle()* sirve para crear líneas sólidas, y *lineGradientStyle()* para aquellas con degradado. Por ejemplo, la siguiente instrucción indica al objeto lienzo, que dibuje líneas de 3 píxeles de grosor, de color rojo dado en notación hexadecimal, y de una opacidad del 75% (100% indica que es totalmente opaco):

```
lienzo.graphics.lineStyle(2,0x990000, 0.75);
```

- **Rellenos:** Para crear una forma con relleno es necesario llamar a los métodos *beginFill()*, *beginGradientFill()* o *beginBitmapFill()*. Explicaremos el más sencillo que es *beginFill()*, cuyos parámetros son: el color de relleno y de forma opcional, un valor *alpha* para éste.

```
lienzo.graphics.beginFill(0x00FF00);//relleno verde sólido
```

Se puede cerrar un relleno con *endFill()*. Si la figura no estaba cerrada, al llamar a este método se dibujará una línea recta entre el punto inicial de la figura y el último para cerrarla. Si no se cierra de forma explícita, al llamar a otro método de relleno, se cerrará el actual automáticamente y se iniciará otro nuevo.

- **Posición de dibujo:** Inicialmente los objetos *Graphics* comienzan su dibujo en el punto (0,0) del espacio de coordenadas del escenario. Para dibujar en otro punto, es necesario llamar primero al método *moveTo()*, antes de emplear cualquier otro método de dibujo.

- **Borrar:** Para borrar del escenario lo incluido en la propiedad, se utiliza:

```
lienzo.graphics.clear();
```

Una vez repasado los pasos previos, se explicará cómo realizar los distintos tipos de dibujo, y que parámetros necesitan con un sencillo ejemplo (todas las distancias se miden en píxeles):

- **Dibujo de líneas:** Para las líneas rectas se usa *lineTo()* pasándole el punto final de la recta. El punto de inicio, será (0,0), o el que se fije con *moveTo*.

```
lienzo.graphics.moveTo(50, 70); //inicio en (50,70)
```

```
lienzo.graphics.lineTo(200, 200); //fin en coordenada (200,200)
```

- **Dibujo de curvas:** Para las curvas se usa el método *curveTo()*. Este dibuja una curva cuadrática de Bézier, es decir, un arco que conecta dos puntos, denominados puntos de ancla, y se inclina hacia un tercero, denominado punto de control. El punto inicial por defecto es el (0,0), a menos que se especifique, por lo que a la función habrá que pasarle las coordenadas del punto de control, y del segundo punto de ancla, en ese orden.

```
lienzo.graphics.curveTo(175, 125, 200, 200);
```

- **Formas comunes:** Las figuras geométricas más comunes disponen de un método que las dibuja de forma automática en ActionScript, que son:

- **Círculos:** Los dos primeros valores corresponde a la coordenada x e y del punto del centro del círculo, mientras que el tercero indica el radio.

```
lienzo.graphics.drawCircle(0, 0, 50);
```

- **Elipses:** El primer par de valores corresponde a la coordenada x e y, de la esquina superior izquierda de la caja de delimitación de la elipse. El tercero indica el ancho, y el último, la altura de la elipse.

```
lienzo.graphics.drawEllipse(0, 0, 60, 70);
```

- **Rectángulos:** La primera coordenada corresponde a la esquina superior izquierda del rectángulo, mientras que los dos últimos son el ancho y alto del rectángulo.

```
child.graphics.drawRect(0, 0, 20, 40);
```

- **Rectángulos con bordes redondeados:** Igual que para los rectángulos normales, pero añadiendo el radio del borde, como último parámetro.

```
child.graphics.drawRoundRect(0, 0, 10, 10, 2);
```

A la hora de incluir un objeto *Graphics* en un objeto de tipo *Sprite* o *MovieClip*, se debe de tener en cuenta, que el contenido de dibujo con la propiedad *graphics*, siempre aparece detrás

de todos los objetos de visualización que estén contenidos en el objeto. No afecta si se han incluido antes o después de añadir el dibujo.

También es importante saber cómo son los ejes de coordenadas, a la hora de dibujar desde código en Flash. Para que las figuras aparezcan en la posición deseada, y no tomen formas inesperadas (lo cual puede ocurrir, sobre todo si se dibujan una serie de elementos de forma automática en un bucle o similar), sus coordenadas se deben calcular teniendo en cuenta que el eje en Flash es:

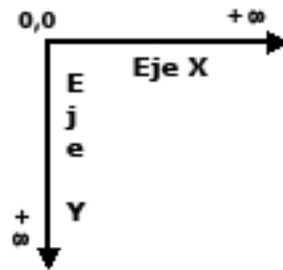


Figura 50. Ejes de coordenadas en Flash.

Por lo que de forma diferente a como se suele representar la zona positiva habitual para los ejes en matemáticas, el eje Y en este caso crece “hacia abajo”. Los elementos que estén fuera de la zona positiva del eje, no aparecerán en el escenario, al igual que aquellos que superen el tamaño en ancho o alto, dado al escenario, al crear un proyecto Flash.

Por último, para manipular el color, se debe de crear un objeto de la clase *ColorTransform* y asignarlo a la propiedad correspondiente del gráfico. Por ejemplo, para asignar un nuevo color de forma aleatoria a una instancia de *MovieClip*, denominada clip, se haría:

```
import flash.geom.ColorTransform;
// Generar valores aleatorios para los canales de RGB
var rojo:Number = (Math.random() * 512) - 255; // aleatorio [-255,255]
var verde:Number = (Math.random() * 512) - 255;
var azul:Number = (Math.random() * 512) - 255;
// Asignar un nuevo objeto ColorTransform con los colores aleatorios.
clip.transform.colorTransform =
    new ColorTransform(1, 1, 1, 1, rojo, verde, azul, 0);
```

Los cuatro primeros parámetros que recibe el constructor de *ColorTransform*, son multiplicadores del valor del color RGB y la transparencia, en ese orden. Por defecto el valor es 1, y el rango va de [0,1]. Los cuatro valores siguientes están por defecto a 0 y van de [-255,255]. Representan la tonalidad del canal de color RGB y la transparencia.

6.2.6 Crear Animaciones Mediante Código. La Clase Tween

Incluir sencillas animaciones utilizando ActionScript, es posible utilizando la clase *fl.transitions.Tween*. Esto permite crear transiciones de movimiento, donde podemos cambiar la posición de un objeto, su tamaño, su rotación, variar la transparencia, o alterar su color, modificando las propiedades del objeto.

Para crear la animación basta con crear una instancia de *Tween*, pasándole los parámetros adecuados a su constructor, que se explican a continuación:

- **obj: Object** - Tipo genérico que indica que cualquier elemento puede ser animado con esta clase. Basta pasar una referencia del objeto, como primer parámetro del constructor.
- **propiedad:String** - Nombre de la propiedad que se verá afectada por la animación (posición, altura, anchura, rotación, alpha).
- **f:Function** - Nombre de la función que va a aplicar el efecto de la animación. Esta función se obtiene importando el paquete *fl.transitions.easing.** y haciendo referencia a alguna de las funciones contenidas en las clases de ese paquete (*Back*, *Bounce*, *Elastic*, *None*, *Regular*, *Strong*).
- **valorInicial:Number** - Valor de inicio de la propiedad que se va a animar.
- **valorFinal: Number** - Valor final que tendrá la propiedad, cuando pase el tiempo fijado para la animación.
- **duración:Number** - Tiempo que va a durar la animación. Si se pone un número negativo o se omite este parámetro, la duración será infinita.
- **usarSegundos:Boolean** - Flag que indica si el tiempo se cuenta en segundos, o en fotogramas. Por defecto está a *false*, a menos que explícitamente se quiera usar en segundos, fijando el valor a *true*.

El siguiente ejemplo hace referencia a la barra de volumen de la aplicación Flash. Esta barra es un clip de película que tiene como nombre de instancia *volumen_mc*, y está compuesto entre otras cosas, por otro clip de película que es una barra que sirve como indicador, con nombre *indicador_mc*. La animación consiste en una ampliación de la barra, respecto a su altura, pasando de un valor de un píxel, a cien, con una duración de medio segundo. El efecto es respecto a la aceleración, que al inicio es mayor, y luego decrece hasta llegar a cero.

```
var myTween:Tween = new Tween(
    volumen_mc.indicador_mc, "height", Strong.easeOut, 1, 100, 0.5, true);
```

6.2.7 Carga de Archivos Externos

Utilizar archivos externos es algo muy frecuente en Flash. Aunque sean imágenes, no siempre se dispone de todas las necesarias en la biblioteca, lo que por otro lado, aumentaría demasiado el tamaño del archivo. Así pues, hay que conocer los medios para cargar ficheros en un momento dado. De forma más avanzada, también se explicará como leer XML, para utilizarlos como archivos de configuración de componentes creados en Flash.

En general, en ActionScript los archivos se cargan mediante la clase *flash.display.Loader*. Para esto, se deben de seguir cuatro pasos:

- Crear un objeto *flash.net.URLRequest* con la ruta del archivo.
- Crear un objeto *Loader*.
- Llamar al método *load()* del objeto *Loader* pasando la instancia de *URLRequest* como parámetro.
- Añadir el objeto *Loader* al contenedor principal, para que se muestre la imagen (se admite imágenes JPEG, GIF o PNG) o el archivo SWF que se quería cargar.

Un ejemplo sería:

```
var request:URLRequest = new URLRequest("ruta/archivo.jpg");
var loader:Loader = new Loader();
loader.load(request);
addChild(loader);
```

En el ejemplo anterior, simplemente se carga un elemento al escenario, y cuando termine la carga, se mostrará (si el archivo es grande, no será algo inmediato, sino que tardará unos segundos).

En caso de querer realizar algún tipo de acción una vez terminada la carga, se tendrán que añadir los eventos necesarios de control. Sin embargo, no se debe poner el evento directamente sobre el objeto *Loader*, sino utilizar la propiedad *contentLoaderInfo*, para realizar el seguimiento a la imagen que se va a cargar, con lo que el ejemplo quedaría de la siguiente forma:

```
var request:URLRequest = new URLRequest("ruta/archivo.jpg");
var loader:Loader = new Loader();
(loader.contentLoaderInfo).
    addEventListener(Event.COMPLETE, completado);
(loader.contentLoaderInfo).
    addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
loader.load(request);
```

Una vez cargada la imagen, en el detector para *Event.COMPLETE*, se podría, por ejemplo, manipular su tamaño, accediendo a sus propiedades a través del objeto de evento. Y mediante la propiedad *loader*, acceder al contenido del archivo y después, añadirlo al escenario. Esto es posible porque el compilador de Flash permite tratar los ficheros cargados de esta forma como si fueran una instancia de *MovieClip*, en lo que respecta a sus propiedades básicas y métodos. Un ejemplo sería:

```
function completado(e:Event):void{
    e.target.loader.width = 200;
    addChild(loader);
}
```

En el caso de querer leer los datos almacenados en un **archivo XML**, el proceso varía un poco. Puesto que hay que esperar primero a la carga completa del archivo para poder acceder a los datos del XML, es necesario incluir un evento que controle cuando termina esa carga. Además, no se utiliza *Loader*, ya que no es algo que se vaya a mostrar en el escenario directamente, sino que se emplea la clase *flash.net.URLLoader* de la siguiente forma:

```
var request:URLRequest =
    new URLRequest("sapientino/pistas_sapientino.xml");
var myXMLLoader:URLLoader = new ULLoader();
myXMLLoader.addEventListener(Event.COMPLETE, cargarXML);
myXMLLoader.load(request);
```

Ahora, en el método detector del evento de carga completa, habrá que tratar la lista de nodos del XML y tratarlos. El siguiente ejemplo muestra como se haría:

```
function cargarXML(e:Event):void {
    var myXML:XML=new XML(e.target.data);
    //se guarda cada nodo PISTA en una posición del xmlList
    var xmlList:XMLList=myXML.PISTA;
    //se muestra la primera pista
    for each (nodo in xmlList) {
        if(nodo.@FRASE != null)
            trace (nodo.@FRASE);//se muestran todas las pistas
    }
}
```

Para poder acceder al XML como pone en el ejemplo, el fichero, tendría que tener una estructura como la que sigue:

```

1 <LISTA intro = "En este juego debes identificar animales, insectos o plantas típi
2   <PISTA ID="rana" FRASE="Anfibio de patas palmeadas. Alternan periodos de vic
3   <PISTA ID="sapo" FRASE="Anfibio de cuerpo rechoncho, ojos saltones y piel gr
4   <PISTA ID="caballito" FRASE="Es un insecto depredador, con abdomen largo, oj
5   <PISTA ID="nenufar" FRASE="Planta acuática de flores blancas."/>
6   <PISTA ID="pato" FRASE="Ave palmípeda de pico chato y patas cortas. Son torp
7   <PISTA ID="zapatero" FRASE="Insecto depredador de antenas largas con pelos c
8   <PISTA ID="espadaña" FRASE="Hierba de tallo largo a manera de junco, cuyo ex
9   <PISTA ID="culebra" FRASE="Es un ofidio. Cuerpo cilíndrico y alargado, cabez
10  <PISTA ID="renacuajo" FRASE="Es una cría o larva, de ranas y sapos."/>
11 </LISTA>

```

Figura 51. Ejemplo de fichero XML.

La clase *XMList* representa un conjunto de elemento XML. Para obtener la información almacenada posteriormente, basta con crear una ruta a partir de la jerarquía de nodos, utilizando el separador ".", teniendo en cuenta que los nombres de nodos se tienen que poner igual que en el fichero XML, y para los atributos de los nodos, estos deben de ir precedidos del signo "@". Se recomienda que se respete el uso de mayúsculas y minúsculas, tal cual se haya escrito en el archivo XML.

A modo de ejemplo, según el archivo de la Figura 51, se podría acceder a la información del *XMList* de varias formas, incluso añadiendo pequeñas condiciones de búsqueda:

```

trace (xmlList[0].@FRASE); // salida: Anfibio de patas palmeadas...
trace (xmlList.(@ID=="pato").@FRASE); // salida: Ave palmípeda...

```

6.2.8 Aplicaciones Distribuidas. Uso de Sockets

Como último punto de este manual introductorio a ActionScript en su versión 3.0, se procederá a explicar cómo funciona la comunicación mediante sockets.

Hay dos clases que ofrecen la base para crear un cliente Flash conectado a su servidor mediante sockets:

- **flash.net.XMLSocket:** Esta clase crea una conexión que permanece abierta hasta que se cierre de forma explícita, de forma que se pueden intercambiar datos XML, sin tener que abrir en cada mensaje nuevas conexiones. Los datos no se solicitan de forma explícita, por lo que se pueden recibir mensajes desde el servidor en cualquier momento.
- **flash.net.Socket:** La diferencia de esta clase con la anterior es que proporciona una forma de leer y escribir datos binarios en cualquier formato. Se pueden usar conexiones de socket binarios para escribir código que permita la interacción con varios protocolos de Internet distintos, como POP3, SMTP, IMAP, NNTP, lo que a su vez permite al reproductor de Flash conectarse a servidores de correos y noticias.

Una vez explicada la diferencia entre ambas clases, se pasará a detallar como realizar una aplicación cliente con sockets binarios, y los métodos necesarios que se deben incluir para su correcto funcionamiento, ya que son los que se han utilizado en la aplicación.

Primero, se tiene que crear una instancia de Socket, para acceder a los métodos de la clase necesarios, de la siguiente forma:

```
var mySocket:Socket = new Socket();
```

Antes de iniciar la comunicación, es conveniente añadir el control de eventos, para por un lado, controlar los posibles errores que se deriven de la comunicación, y por otro lado, activar el mecanismo de seguimiento de todos los mensajes entrantes que puedan llegar por el socket. Por lo que se añaden las siguientes instrucciones para los tipos de evento:

- **Event.CLOSE:** Este evento aparece en los casos en los que pierda la conexión por socket, o el servidor lo haya cerrado.
- **IOErrorEvent.IO_ERROR:** Este evento contempla los casos en los que el cliente no es capaz de realizar la conexión.
- **SecurityErrorEvent.SECURITY_ERROR:** Este error se produce al tratar de acceder a una IP no accesible en la red.
- **ProgressEvent.SOCKET_DATA:** Evento que se recibe cuando llega un mensaje.

Las instrucciones serían:

```
mySocket.addEventListener(Event.CLOSE, controlErrores);
mySocket.addEventListener(IOErrorEvent.IO_ERROR, controlErrores);
mySocket.addEventListener(
    SecurityErrorEvent.SECURITY_ERROR, controlErrores);
mySocket.addEventListener(ProgressEvent.SOCKET_DATA, leer);
```

Después, se hace la llamada al método *connect*, para realizar la petición de conexión al servidor. Para ello se tiene que pasar como parámetro el host, que será la IP del equipo que contenga el servidor al que se va a conectar el programa, y un número de puerto, que coincida con el puerto abierto habilitado para escuchar los mensajes entrantes y las peticiones de conexión. La IP debe ser de tipo String, y el puerto, un número.

```
mySocket.connect(host, port);
```

Una vez se ha llegado a este punto, y no se detecta que se haya producido un error, al cabo de unos segundos, es que la conexión se ha realizado con éxito.

Cuando la comunicación ya se ha establecido, el siguiente punto necesario es procesar los datos que se van a enviar y recibir.

Para leer un mensaje del socket, en la función detectora del evento, se incluirá la siguiente instrucción:

```
var str:String=mySocket.readUTFBytes(mySocket.bytesAvailable);
```

Esta línea de código permite leer los datos de bytes en formato UTF-8 que han llegado por el socket, aplicando un formato de cadena de caracteres, y guardarlos en una variable de tipo String. A partir de ahí, se podrá tratar el mensaje como texto normal.

La cantidad de datos se limita hasta el primer salto de línea que aparezca en los datos recibidos. El número exacto se obtiene a partir de la propiedad *bytesAvailable* de la clase *Socket*, que devuelve el número de bytes que están a la espera para poder ser leídos en el buffer de entrada del socket.

Ya que esta acción se realiza de forma inmediata únicamente al detectar la llegada de datos por el socket, y cuando se envían mensajes de texto desde Maggie, tampoco habrá problemas de que se reciba menos de un byte, y el control de excepciones en esta parte del código no es necesario.

Para enviar un texto por el socket, se emplea el mismo formato UTF-8 que en la lectura, pero utilizando el método que realiza la operación inversa, pasando de texto a datos binarios. Se debe de incluir el carácter de fin, que en Flash se representa con "0", para delimitar el tamaño

del mensaje, y se reciba correctamente en el servidor. Una vez escrito todo lo necesario en el buffer, se vuelca en el socket, con la orden *flush()*.

El código completo es el siguiente:

```
try {  
    mySocket.writeUTFBytes("texto");  
    mySocket.writeByte(0);  
    mySocket.flush();  
} catch (errObject:IOError) {  
    trace("Error:" + errObject.message);  
}
```

Las instrucciones de lectura deben de encapsularse en una sentencia de try-catch, debido a posibles excepciones que pueden aparecer si por ejemplo, el socket se ha cerrado (esto se podría controlar específicamente con la propiedad booleana *connected* de la clase *Socket*), y aún así se trata de enviar información, o por no poder escribir correctamente en el socket. De esta forma se controla el error para no terminar de forma abrupta la aplicación.

Y para cerrar la comunicación del socket, se emplea el método:

```
mySocket.close();
```

6.3 Manual de la Aplicación

En esta sección, se tratará de explicar en detalle, como funciona la aplicación Flash desarrollada en este proyecto. Por un lado se expondrá como se ha hecho cada punto, para que en caso de realizar modificaciones que pueden ser necesarias en un futuro, se hagan de forma sencilla. Y a su vez, también se mostrará como añadir nuevos elementos similares a los ya incluidos en la interfaz.

6.3.1 Editar la Estructura General

Para comenzar, lo más básico es saber cómo está dividida la interfaz en los distintos fotogramas y capas.

La línea de tiempo principal del proyecto, consta de catorce fotogramas y, nueve capas (ver Figura 52). Cada una de estas capas contiene objetos que afectan a uno o varios fotogramas, y se han dividido siguiendo el mismo criterio que con las secciones, de forma que hay una capa preparada para contener los objetos de visualización de cada sección.



Figura 52. Línea de tiempo del proyecto.

El contenido de cada una de las capas, es:

- *Display de estados:* Los elementos incluidos sólo se muestran en el fotograma número tres, que es el que se utiliza para mostrar la sección del Display de Estados. Los gráficos que corresponden a los ejes y el fondo del Display, están aquí.
- *Menú:* Los botones para pasar de página, en caso de que la lista de botones del menú de eventos, no quepa en una ventana, se incluyen aquí. Estos símbolos se ven en el fotograma número cuatro.
- *Música:* Contiene los botones de control, el componente de lista, los campos de texto, la barra deslizador, y los campos donde se mostrará el tiempo del reproductor de audio que se muestra en el fotograma número cinco.

- *Imágenes*: Como su nombre sugiere, la galería de imágenes se corresponde con esta capa. Se aplica al séptimo fotograma, donde se mostrarán los botones de la navegación interna de la galería, para el conjunto de imágenes en miniatura, y la barra que separa la imagen ampliada, del lienzo de miniaturas, además de los campos de texto.
- *Botones*: Incluye los botones propios de la navegación interna de la interfaz, así como el botón con el logotipo del laboratorio de robótica, por lo que tiene elementos en los catorce fotogramas que se utilizan de la línea de tiempo principal. Los botones que controlan secciones específicas se han incluido en la capa de la sección correspondiente.
- *Bubbles_tablero*: Envuelve la instancia del clip de película que representa el tablero del juego Bubbles, y el texto que se añade para explicar el juego, y avisar del fin del mismo. Para dividir las partes entre la introducción, el juego en sí, y el final, se utilizan los fotogramas del nueve, al once.
- *Sapientino_laguna*: La imagen de la laguna, los botones de los nombres y de los animales, además del botón de pistas, y los textos al inicio y al finalizar mostrando las puntuaciones, están dentro de esta capa. Ocupa del fotograma doce, al catorce.
- *Fondo*: Los elementos gráficos que comprende, son los que estarán debajo del todo, para dejar que los de las demás capas se vean por encima. Se ha añadido una base con aspecto de marco, para la aplicación. Los textos de inicio de las secciones, y algunas imágenes que se usan de marca de agua, o la animación de cielo estrellado que hace de fondo del juego Bubbles, también se incluye aquí.
- *Script*: esta es una capa especial, donde sólo hay una única instrucción `ActionScript stop()`, que está presente del primer al último fotograma que haya con gráficos de la aplicación. De esta forma, se evita que Flash intente reproducir todos los fotogramas seguidos, para que sea el programador el que controle la navegación.

La ventana de inicio de la aplicación corresponde al fotograma uno, mientras que el menú principal ocupa el número dos. A partir de ahí están las distintas secciones en el orden en el que aparecen en el menú. Después se colocó el menú de juegos en el fotograma ocho, con cada uno de los juegos, de forma secuencial, por detrás. Se ha tratado de dar un orden de forma que a la hora de implementar la parte de la navegación, los números de fotogramas fueran sencillos de recordar, para saber a qué fotograma había que mover la línea de tiempo, al activar un botón u otro.

Respecto al sexto fotograma de la línea de tiempo, corresponde a la sección del reproductor de video. Ya que todos los componentes de este apartado se crean en el código, no era necesario añadir una capa más por esto. Lo único que se modifica es el nombre de la sección, y eso se considera parte del fondo.

Si se quieren añadir secciones en algún momento, se recomienda utilizar capas nuevas para los elementos de ese apartado, y utilizar un fotograma libre donde mover la cabeza lectora, cuando se elija esa opción en el menú.

Si se incluyen más juegos, estos pueden colocarse a partir del fotograma número quince.

En caso de secciones accesibles desde el menú principal, se podrían desplazar los fotogramas correspondientes a los juegos (pero esto implica el tener que reconstruir la navegación interna en el código), o simplemente poner al final todo lo que se tenga que añadir a la interfaz.

Si la aplicación se vuelve muy compleja, se puede recurrir a nuevas escenas en las que añadir el código preciso, sin tocar la disposición actual, según las preferencias del desarrollador.

Finalmente, recordar que la capa Script debe alargarse hasta cubrir todos los fotogramas con elementos, para seguir utilizando cada uno de ellos como un apartado diferente, para que no se muestren como una animación, todos seguidos. Esta capa, suele permanecer la última, y sin elementos gráficos, ya que se reserva para el código.

Respecto al tipo de letra que predomina en todos los títulos, y en las etiquetas de los botones de los menús, el estilo utilizado es *Jokerman*. Para el resto de textos, se ha utilizado la tipografía *Arial* para no recargar el estilo final de la interfaz. Ésto se puede modificar en el inspector de propiedades de cualquier campo de texto.

En cuanto al tamaño de la letra, se ha tratado de ajustar en cada caso, para colocar textos grandes, que puedan leerse a cierta distancia, sin que estos lleguen a predominar frente al resto de elementos de la pantalla.

Si se quiere sustituir algún botón de los utilizados, por otros que se consideren más atractivos visualmente, sin tener que modificar el código interno, basta con mantener el nombre de la instancia dado a los botones actuales, en los nuevos. En caso de botones como el del volumen del reproductor de música, que a su vez contiene instancias de otros elementos a los que se hace referencia desde el código, también se tendrían que mantener esos nombres, o realizar las modificaciones pertinentes en las clases afectadas, para que continúe funcionando la aplicación.

Pasando al tema del fondo, está compuesto por dos rectángulos superpuestos, con bordes de trazo grueso y punteado, y rellenos de dos tonos de azul. Esto aporta la sensación de que la interfaz estuviera en un sencillo marco, y el estilo del borde le da un estilo informal. Para modificarlo, se tienen que editar o sustituir los dos rectángulos en la capa *fondo*. Sin embargo, las dimensiones y la posición del recuadro más grande deberían respetarse. Esto se debe a que la interfaz está colocada con exactitud en el hueco que hay en la carcasa de Maggie, teniendo en cuenta la posición y tamaño del escenario cuando se ejecuta el programa en pantalla completa. Por lo tanto, para evitar tener que realizar sucesivas pruebas de reajuste, las medidas adecuadas y la posición de la zona que quedará visible posteriormente en el tabletPC son las siguientes:

- **Alto:** 890.1
- **Ancho:** 638.2
- **Posición X:** 0
- **Posición Y:** 44.9

Las medidas anteriores son las adecuadas para ajustar la interfaz, cuando el tamaño del escenario del proyecto es de 696x975 píxeles. Este valor se ha calculado de forma proporcional al hueco de 15x21 cm del pecho de Maggie.

Para futuras modificaciones, se ha de tener en cuenta, que el ajuste no es trivial, ya que el hueco no es totalmente rectangular, sino trapezoidal, además de que está ligeramente desplazada la pantalla del portátil, y ajustar el alto del escenario, al colocar en pantalla completa, también se ajusta de forma proporcional el ancho. Con lo que realizar una proporción matemática no es exacta, y el método de prueba y error puede resultar bastante tedioso, entre otras cosas, por la lentitud de la conexión de escritorio remoto con el tabletPC, ya que se necesita publicar el archivo Flash a *.exe con cada modificación, para probar la posición activando la pantalla completa.

6.3.2 Cómo Enviar/Recibir Información de Maggie

El código del cliente, implementado tal como se indica en el punto 6.2.8 Aplicaciones Distribuidas. Uso de Sockets, se ha incluido directamente como parte de la clase principal del programa (Main.as).

La conexión se realiza en respuesta al evento *MouseEvent.CLICK*, sobre el botón “Conectar”, de la ventana de inicio (ver Figura 11). Si la conexión no se ha realizado con éxito aparecerá un mensaje de error para avisar al usuario (ver Figura 12). Esto no impide poder utilizar algunas partes del programa como los juegos, la galería, y los reproductores de audio y video. Sin embargo, no existirá respuesta del robot en el display, el menú de eventos, ni habrá comentarios de Maggie en el transcurso de los juegos.

Una vez realizada la conexión, se pueden enviar mensajes en cualquier momento, haciendo una sencilla llamada a:

```
public function enviar(cadena:String):void{...}
```

Este método se encuentra en el Main, por lo que mediante un objeto *m* que haga referencia a la clase principal, se podrá acceder desde cualquier punto de la arquitectura, ya que la visibilidad del código es *public*, con:

```
m.enviar("texto del mensaje");
```

En cambio, los mensajes entrantes necesitan algún tipo de respuesta explícita en el código, por lo que su tratamiento no puede ser totalmente transparente al usuario como al enviar.

Al realizar la petición de conexión, se permanece a la escucha del evento que avise de la llegada de datos por el socket. Cuando este evento se activa, se llama a la siguiente función del Main:

```
private function leer(event:ProgressEvent):void {
    //Obtener mensaje
    var str:String=mySocket.readUTFBytes(mySocket.bytesAvailable);
    //Se divide en campos
    var miStringArray:Array=str.split(":");
    //Los mensajes del display, sólo se admiten en el fotograma 3
    if (currentFrame==3 && miStringArray[0]=="display") {
        //Se actualiza el display
        display0.actualizar(parseInt(miStringArray[1]));
    }
}
```

Como se puede ver en el código, esta es una función *private*, lo que denota que es propia del Main exclusivamente. La idea es, que en este método se leen los datos del socket convirtiéndolos en texto, se divide en trozos utilizando el separador “:”, y se identifica que mensaje es, utilizando como base el protocolo de mensajes (ver Tabla 30. RNF-13. Protocolo de mensajes). El resultado se asigna a un array, y accediendo a la posición adecuada, se obtendrá el elemento que se necesite. Ejemplo:

```
var mensaje:String = "display:50:24";
var array:Array = mensaje.split(":"); //quedaría {display, 50, 24}
var i:Number = 0;
for (i; i<array.length;i++){
    trace ("Elemento en la posición " + i + ": " + array[i]);
}
// Resultado mostrado en pantalla
//Elemento pos 0: display
//Elemento pos 1: 50
//Elemento pos 2: 24
```

Todos los elementos del array serán de tipo String, por lo que en caso de querer cambiar el tipo de los valores a números, se tendría que utilizar el método:


```
parseInt ("número"); //devuelve un valor entero
parseFloat ("número"); //devuelve un valor decimal
```

De esta forma se obtiene la cabecera del mensaje, y a partir de ella, con bloques *if*, se puede averiguar qué mensaje se ha recibido y actuar en consecuencia.

Actualmente, sólo se tienen en cuenta los mensajes recibidos cuando la interfaz se encuentra en la sección del display de estados, que corresponde al fotograma número 3. Se comprueba que la cabecera del mensaje sea “*display*”, por lo que se tendrá que realizar la llamada para actualizar el gráfico, a partir de la instancia que se habrá creado de esa clase, al entrar en la sección.

Se espera que poco a poco, se vayan añadiendo más mensajes, de forma que se tendrá que añadir una sentencia condicional nueva que contemple ese caso y, tratarlo de la forma correspondiente.

Para no complicar excesivamente el método, y facilitar la comprensión del flujo de acciones, se aconseja añadir una única instrucción en respuesta a cada estructura condicional. En caso de que se tengan que realizar un número elevado de pasos, será mejor encapsular ese código en una función que sirva de intermediaria, y que pueda invocarse desde el Main.

Para finalizar la comunicación, esto se hace cuando se cierra la aplicación, al pulsar en el botón , que se encuentra en el menú principal (ver Figura 14). De esta forma, si se había producido la conexión correctamente, se enviará un mensaje al servidor, con una única palabra “*cerrar*”, para avisar del fin de la comunicación, y se proceda al cierre del socket, adecuadamente.

6.3.3 Cómo Usar/Modificar el Menú de Eventos

La configuración básica del menú de eventos se hace a través de un fichero XML. En este archivo se incluye toda la información necesaria para enviar los eventos de activación y desactivación de cada habilidad, además de otros datos referentes a los botones e iconos a utilizar.

Los nodos principales del archivo XML tienen la etiqueta “Evento”, y un atributo “id”. El primer nodo varía con respecto a los demás, ya que se emplea para saber si se va a usar el modelo de un único botón (ver Figura 15), o el modelo de dos botones partidos (ver Figura 16). Para diferenciarlo en el código, se le ha asignado como atributo “id” la palabra *formato*, de forma que en ese caso, el nodo tendrá una única etiqueta hija denominada “numBotones”, donde habrá que poner un “1” o un “2”, según el caso.

```
<Evento id="formato">
    <numBotones>2</numBotones>
</Evento>
```

Cada uno del resto de los nodos, hará referencia a una habilidad de Maggie, que se verá representada por un botón en el menú de eventos. Los botones se añadirán en el orden en el que se lean del fichero, y para poder tener un control sobre su organización, el “id” será un número entero secuencial que comenzará en cero. Este valor se utilizará en el código, para identificar cada botón y controlar si se ha pulsado o no, en el caso del modelo de botón único.

A continuación se incluye un ejemplo de los datos a incluir para el caso de la habilidad de pronóstico del tiempo:

```
<Evento id="0">
    <eventoStart>10074</eventoStart>
    <paramStart>0</paramStart>
    <eventoStop>10075</eventoStop>
    <paramStop>-1</paramStop>
    <texto>Tiempo</texto>
    <icono>files/icsol.png</icono>
    <frase>¿Qué tiempo hará hoy?</frase>
</Evento>
```

Los nombres de las etiquetas se deben respetar rigurosamente. De esta forma se incluye el identificador de los eventos, los parámetros que se deben de enviar en cada caso (si no es necesario enviar ningún parámetro, se deberá poner un “0”), el texto que aparecerá identificando el botón (independientemente de cómo se escriba en el XML, en la interfaz saldrá en mayúsculas), y la ruta del icono que se mostrará al lado del botón (si no se quiere añadir un icono, no debe haber nada entre las etiquetas “icono”). También se ha incluido la etiqueta

“frase”, para el caso en el que interese hacer que Maggie diga algo, junto con la realización del evento que se seleccione en la interfaz. Esta frase deberá estar escrita en el fichero en español o en inglés, según la versión de la aplicación con la que se esté trabajando.

Un ejemplo de cómo sería el botón dibujado en la interfaz, con los valores de ejemplo sería:

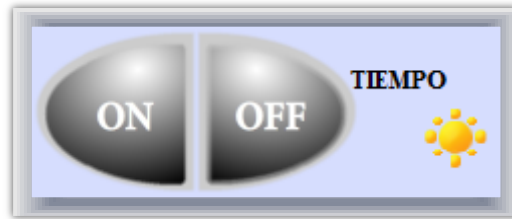


Figura 53. Ejemplo de botón del menú de eventos.

Hay cuatro clases diferentes asociadas a los distintos botones que pueden aparecer en este menú. Como ya se ha explicado, debido a que el texto que se incluye en los botones debe de añadirse como campo de texto a su línea de tiempo, para que no aparezca una zona en la que no responda a la interacción con el ratón, se tienen que diseñar tanto el botón con el “ON” (activar habilidad), como el botón del “OFF” (desactivar). Y para cada par, está el modelo unido o con los botones por separado, lo que hace las cuatro clases diferentes de las que se hablaba, que son:

- *BotonUnidoON.as*
- *BotonUnidoOFF.as*
- *BotonPartidoON.as*
- *BotonPartidoOFF.as*

Todas ellas están en el paquete *com*.

En caso de que se quieran modificar los botones que se han empleado, bastaría con asociar estas clases *ActionScript* a los nuevos diseños, dentro de la biblioteca del proyecto de *Flash*, tal y como se explica en el punto 6.1.8 Como Asociar Ficheros de *ActionScript* al Proyecto *Flash*.

El código de cada una de estas clases es muy sencillo. Comenzamos a explicar los elementos del constructor:

- En el caso del modelo unido, lo que recibe por parámetro es lo mismo en los dos casos (ON y OFF):
 - El contenedor del botón, de tipo *Sprite*, donde añadir el texto y el icono.
 - Una variable de tipo *MenuDinamico*.
 - La posición x e y donde se ha de colocar, ambas tipo *Number*.
 - El identificador dentro del código, de tipo *String*.
 - El número del evento, en formato *String*.

- El parámetro que le corresponde al evento, también en formato *String*.
- El texto que le identificará en la interfaz. Tipo *String*.
- La ruta del icono que se colocará junto al botón. Tipo *String*.
- La frase que se incluye en caso de que Maggie vaya a decir algo, en *String*.

Todos estos elementos se tratan en el constructor, de forma que fija la posición del botón según los valores dados, y se añade en un campo de tipo *flash.text.TextField*, el texto y el icono, a una distancia suficiente para que quepan dos botones por cada fila, se añade a la lista de visualización, y se incluye el evento de clic de ratón.

Además de esto, se incluye el método *enviar*, que es la función manejadora del evento. Como su nombre indica, se encarga de mandar la cadena de texto apropiada con el número de evento y el parámetro, al servidor en VisMaggie. Para ello no llama directamente a la función del Main, sino a una intermediaria de la clase *MenuEventos*. También se hace una llamada a la función *swichtON* ó *swichtOFF*, según lo que corresponda, que elimina el botón actual, y posiciona en el mismo sitio, el botón con la orden contraria. Y en caso de querer enviar un mensaje para hacer hablar a Maggie, con la frase incluida en el XML, se especificaría aquí.

- En el caso del modelo Partido, varía un poco la funcionalidad:
 - Botón ON: La funcionalidad es muy similar a la explicada anteriormente. Recibe los mismos parámetros que ya se han detallado, y en el mismo orden, a excepción del identificador, el texto y la ruta para el icono, por lo que los pasos que se refieren a estos elementos no se realizan.

En cuanto a su método *enviar*, sirve para mandar el evento de activar una habilidad, pero esta vez, como ya están incluidos los dos botones, no se alterna entre uno y otro, por lo que esa instrucción no es necesaria.

- Botón OFF: El constructor es algo más largo, que en el caso del ON. Esto se debe, a que sí se reciben por parámetro los mismos elementos que en las clases de los botones unidos, a excepción del identificador, que no hace falta. La diferencia con el botón ON, radica en que al ir ambos juntos, el texto y el icono sólo se asocian a uno de los dos botones. Ya que el botón OFF se coloca a la derecha, y por lo tanto, es el más cercano a estos elementos, es más fácil colocar su posición tomando como referencia este botón, y no su compañero.

Respecto al método *enviar*, en este caso sirve para mandar el evento de apagado de una habilidad.

Los mensajes de voz, en ambos casos se realizarán al enviar.

En cuanto al flujo de datos de la clase *MenuDinamico*, esta clase se encarga de colocar los botones por el escenario de forma automática y ordenada, según los datos del XML, y añadir una navegación interna que permita el desplazamiento del usuario. Si la detallamos por funciones, sería:

- **Constructor:** se recibe como parámetro una instancia de la clase *Main*, para poder acceder a la lista de visualización, y poder enviar datos al servidor. Dentro del constructor se procede a leer el archivo XML, y se inicializan varios atributos:

```
pagina = 1;
listaActivados = new Array();
contenedor = new Sprite();
```

- *página:* Sirve para saber la “página” actual de la lista de botones, útil en el caso de que no se puedan mostrar todos en una misma ventana (el máximo son seis, y esto se controla con el atributo *BOTONESPORPAG*).
- *listaActivados:* Esta variable es de tipo array, para guardar una lista de valores booleanos, donde en el caso de los botones unidos, se almacenará si un botón se ha pulsado o no, para recordar su estado a la hora moverse de forma interna, por la lista de eventos, cambiando de “página”.
- *contenedor:* Se utiliza para agrupar todos los botones, iconos, y campos de texto en un mismo sitio, de forma que al pasar de página, se puedan borrar todos con comodidad, y volverlos a añadir. Este Sprite, se añade como elemento hijo del *Main*.

Para leer el XML, se utiliza la clase auxiliar *cargaXML*. Está en una clase separada, para poder ser analizada y reutilizada más fácilmente, como ejemplo para aprender a manejar XML en Flash. De esta forma, la clase *MenuDinamico* se abstrae de cómo se lee el fichero, y simplemente crea una instancia de esta clase auxiliar, pasando una referencia del menú, para poder continuar la ejecución, cuando se haya completado la carga del fichero.

```
var xml:cargaXML = new cargaXML(this);
```

De forma resumida (para más detalle ver 6.2.7 Carga de Archivos Externos), dentro del constructor de la clase *cargaXML*, se lee el fichero a partir de la ruta del mismo (*eventos.xml* está en paquete *menú*, junto con las clases *.as) y se añade el evento de control de fin de lectura del fichero. En la función manejadora de este evento, se recorren los nodos, y se guarda la información en una cadena de texto. Datos de nodos diferentes, se separan con el carácter “\$”, y la información de cada elemento del nodo con “:”. Finalmente, se llama al método *procesarDatosXML*, de *MenuDinamico*, pasando el número de botones, y la cadena de texto.

- **ProcesarDatosXML:** Esta función recibe los datos del XML, a partir de los cuales, calculando el número total de botones, y de páginas, activando en caso de necesidad los botones *siguiente* y *anterior*, además de los botones que se incluyen para volver al menú principal, y para poner en pausa la aplicación.

El atributo *listaActivados* se inicializa por defecto a 1 (ON =1, OFF=0), sólo en caso de que se utilice un único botón, ya que para dos, no es necesario guardar su estado.

Por último, se realiza una llamada al método *pintarBotones*:

```
pintarBotones(cadena, 0,
              Math.min(totalBotn, pagina*BOTONESPORPAG/numBotones));
```

- **PintarBotones:** Este método es el encargado de añadir al escenario los botones correspondientes a la página actual, en base al límite inferior y superior que recibe por parámetro, y colocarlos en las posiciones adecuadas para que no se solapen unos con otros.

Los parámetros en el caso anterior, indican:

- La cadena con todos los datos del XML.
- El límite inferior, que representa el identificador del primer botón que hay que añadir en la página actual (en el caso de la primera página, el identificador será cero, al pertenecer al primer elemento de la lista).
- El límite superior, controla el último botón que debe de aparecer en la página actual. Se hace el mínimo entre el número total de botones, y el valor del identificador que tendría que tener el último botón, en caso de que la página estuviera completa (si como máximo son seis por página, pero sólo hay cuatro habilidades, el límite superior es cuatro, y no seis). Se multiplica por *numBotones*, porque en el modelo partido, habrá el doble.

En esta función se separan los elementos de la cadena utilizando *split*, y se recorren con un bucle, de forma que en cada iteración se crea y añade un botón del modelo que corresponda (comprobando si está activo o no), al atributo contenedor. Se lleva la cuenta de filas y columnas (dos botones por fila en el caso de los unidos, o cuatro, si es partido), y en base al lugar (fila y columna), que le toque al botón, se le asigna un valor para sus coordenadas x e y, de forma que se colocan automáticamente por el escenario. Por lo tanto, en caso de querer modificar la separación entre los botones, se debe ajustar aquí.

```
var bt1b:BotonUnidoOFF = new BotonUnidoOFF( contenedor, this,
      65+contadorFilas*290, 250+contadorColumnas*150, array[0],
      array[3], array[4], array[5], array[6], array[7]);
```

El valor que se suma, es la posición mínima x o y donde se encontrará la figura, y el que se multiplica por la fila o la columna, es la separación horizontal y vertical entre botones.

El resto de parámetros que se pasan al constructor, son los obtenidos de la cadena de texto, una vez hecha la separación con *split*.

El resto de funciones de la clase *MenuDinamico*, son para alternar entre los botones ON y OFF, del modelo unido, y para realizar la navegación interna. Seguidamente se explica su funcionamiento:

- **swichtON/swichtOFF:** Este método recibe por parámetro el identificador del botón, que indica la posición en el array *listaActivados*, donde registrar el nuevo estado del botón. Una vez hecho esto, se borrará del escenario el botón anterior, creando después uno nuevo, con la orden contraria, y añadiéndolo al escenario. Al constructor del nuevo botón, se le asignará la misma posición del antiguo, y se obtendrán de nuevo los parámetros de la cadena de caracteres del XML, ya que los valores del evento cambian.
- **paginaAnterior/paginaSiguiente:** Estas funciones responden al evento de clic de ratón sobre los botones *<anterior y siguiente>* de la interfaz. Los pasos a seguir son:

- Se quita *contenedor* del escenario y se instancia de nuevo, para borrar todo lo almacenado.

```
m.removeChild(contenedor);
contenedor = new Sprite();
```

- Se actualiza la página según si se ha pasando hacia delante o hacia atrás, y se llama de nuevo al método *pintarBotones*, con la siguiente fórmula para obtener los límites adecuados en cualquier caso:

```
pintarBotones(cadena, limiteInferior, limiteSuperior));
```

El límite inferior debe ser cero para la primera página, y como la numeración de las páginas empieza en 1, se resta uno, y se multiplica por el número de botones por página (*numBotones* sirve para multiplicar por dos, en caso de que se usen los botones partidos). Como todas las páginas se rellenan al completo, el límite inferior en el caso de páginas siguientes, será el máximo de botones por página, por el número de páginas, lo cual queda en la fórmula genérica:

$$(pagina-1) * BOTONESPORPAG / numBotones$$

El límite superior, de forma similar a lo que ya se explicó en el caso base (para la primera página), ahora debe incluir la página actual en la fórmula, porque el máximo será o bien el total de botones, o el máximo número de botones que quepan, teniendo en cuenta que todas las páginas hasta la actual (incluida) se llenan al completo (si el total son 15, en la segunda página, el límite superior será 12, y no 15, y en la tercera, será 15 en lugar de 18 que sería el máximo si se colocasen 6 botones por página). La fórmula queda:

$$\text{Math.min}(\text{totalBotn}, \text{pagina} * \text{BOTONESPORPAG} / \text{numBotones})$$

- Una vez creados todos los botones, se añade el contenedor actualizado de nuevo al escenario, para que se muestre


```
m.addChild(contenedor);
```
- Por último, se controla si se ha llegado a la última página (o a la primera), para desactivar (o activar, suscribiéndose de nuevo al evento de clic), el botón correspondiente, de forma que no se pueda seguir avanzando (o retrocediendo).
- **Enviar:** Cuando se pulsa un botón, se realiza una llamada a este método que accede a la función del mismo nombre del Main, donde se manda la información al servidor. Sirve de puente entre las clases, para seguir un orden lógico en el flujo de la ejecución, teniendo en cuenta la relación entre las clases.

En caso de que el usuario pulse para volver al menú principal, se borra del escenario el contenedor de los botones y gráficos, antes de cargar el fotograma y los eventos correspondientes al menú principal.

6.3.4 Cómo Usar/Modificar el Display de Estados

La clase que controla esta sección (ver Figura 17), es *DisplayEstados.as* del paquete *display*.

Se ha implementado de forma que el control sobre la posición, y el contenedor que se use como contexto para el gráfico, se definan fuera de la clase. Cada display que se desee mostrar, se tendría que inicializar de forma análoga a las instrucciones contenidas en el método *initDisplay* del *Main*, donde *clipDisplay* y *display0* son atributos de la clase principal:

```
private function initDisplay():void {
    //La posición "x" e "y" coincide con el eje en el dibujo
    clipDisplay = new MovieClip();
    clipDisplay.x = 275+25; //se añade un margen derecho del eje
    clipDisplay.y = 455.5;
    display0=new DisplayEstados(this,clipDisplay,18,100);
    addChild(clipDisplay);
}
```

Así, no será necesario modificar internamente el display, y bastará con conocer tres instrucciones para usarlos:

- Una instancia donde se invoque al constructor de la clase, pasando una referencia de la clase *Main*, un contenedor de tipo *MovieClip*, y dos números: el ancho de la barra del display, y su altura inicial. Una vez hecho esto, bastará con añadirlo a la lista de visualización para que se muestre con el valor de inicio que se le haya dado.

```
display0=new DisplayEstados(this,clipDisplay,18,100);
```

- Una llamada al método *actualizar*, pasando como parámetro el valor numérico en el rango (-100,100), a partir del cual, se calculará el tamaño de barra del gráfico que se muestra en pantalla. Esta sirve para los posteriores cambios de valor, ya que en el constructor se asigna por primera vez la altura. Borrará y creará de nuevo el gráfico.

```
display0.actualizar(52);
```

Si el rango es distinto, se deberá convertir el que se usa en la clase, para que el display que se muestre sea proporcional.

- La sentencia que ejecuta la función *stop*, que borra del escenario el display que hubiera dibujado. Se usa al cerrar esta sección y volver al menú principal.

```
display0.stop();
```

En principio, y dado que se pretenden utilizar varios gráficos a la vez, la altura necesaria del eje del display es de 150 píxeles (contando el trozo de barra para valores positivos y para los negativos). En caso de querer ampliar, o reducir el tamaño de la barra que se muestra, se tendría que hacer en el código de la clase *DisplayEstados*. Hay dos formas posibles:

- Si se quiere hacer más grande, o más pequeña, pero respetando la transición de color actual, para el rango de valores (-100,100), se deberá modificar el atributo *ALTOPORCIONBARRA*. De momento, esta constante tiene un valor de 7, lo que implica que cada porción de un color diferente, tendrá una altura de 7 píxeles.
- En la función privada *getColor*, a partir de la altura, se devuelve una posición del array *colores*, que a la hora de pintar la barra, servirá para saber cuántos fragmentos se deben colocar para que el gráfico tenga el tamaño deseado. Si se quieren añadir más porciones de colores, o cambiar el rango de valores, se deberá hacer en este método, modificando los intervalos para que contemplen un rango distinto del actual, o añadir más colores a la lista.

De momento se ha incluido una gama de colores, en sus códigos hexadecimales siguiendo el formato RGB, que empieza con verde (posición cero de *colores*), hasta el rojo (última posición), pasando por el amarillo, de forma que la tonalidad va cambiando de uno a otro, poco a poco.

El display se dibuja en *start(altura:Number)*. Según el valor que se dé, se creará una barra con un número de porciones proporcional al número dado. Este método privado se llama desde el constructor o al actualizar (en el segundo caso, se hará primero una llamada a *stop*). Su función consiste en:

- Distinguir si la barra es positiva o negativa, para saber si se debe de dibujar hacia arriba o hacia abajo.
- Recorrer en un bucle las posiciones de la lista *colores*, hasta la posición obtenida al invocar *getColor()*, pintando en cada iteración una porción de la barra, del color que corresponda. Los trozos de cada color, son rectángulos dibujados con la clase *Graphics* (ver el punto 6.2.5 Crear Formas Gráficas desde Código. Uso de la Clase *Graphics*). Las coordenadas para indicar el tamaño, se calculan automáticamente en base al número de la iteración del bucle, el ancho de la barra (asignado en el constructor), y con la variable *ALTOPORCIONBARRA*. La siguiente línea es un ejemplo para el caso de que la barra sea negativa.

```
ref.squareN.graphics.drawRect(
    x, y +(i*ALTOPORCIONBARRA), anchoBarra,ALTOPORCIONBARRA);
```

6.3.5 Cómo Usar/Modificar la Galería de Imágenes

La galería de imágenes (ver Figura 25), obtiene la lista de elementos que mostrar a partir del fichero *verMiniaturas.xml*, que se encuentra en la carpeta *galeriaImagenes*, en la que se incluye, por un lado, la ruta donde encontrar las imágenes en miniatura que servirán para la vista previa, y por otro, la ruta de las imágenes en su tamaño original, para verlas ampliadas. Estos dos datos se obtienen a partir del atributo “carpetaminis” y “carpetagrandes”, del nodo principal “RUTAS”.

A partir de ahí, se incluye un nodo con la etiqueta “ARCHIVO”, por cada una de las imágenes de la lista, especificando su nombre (y su extensión), como atributo del mismo, tal y como aparece en el siguiente ejemplo:

```
<RUTAS carpetaminis="galeriaImagenes/minis/"
      carpetagrandes="galeriaImagenes/originales/">
  <ARCHIVO nombre="ave.jpg"/>
  <ARCHIVO nombre="calculadora.jpg"/>
  ...
</RUTAS>
```

El tamaño de las imágenes debe de ser, como máximo, de 100x100 px las miniaturas y de 315x315 px las originales. Los marcos de las imágenes, la posición centrada en el escenario y la distancia de separación entre ellas, se ha calculado para estas medidas. Además de esto, el nombre de una imagen debe ser igual tanto para el fichero de la carpeta *minis*, como *originales* y se debe de corresponder con el nombre de cada nodo “ARCHIVO”.

El código que controla la funcionalidad de esta sección está en el paquete *galeriaImagenes*, y está formado por las clases *ImagenesMini.as* y *GaleriaImagenes.as*.

Las imágenes están distribuidas en distintos contenedores (todos ellos de tipo *Sprite*), para facilitar la tarea de actualizar los elementos que se muestran. Por un lado está *contenedor_principal* que contiene todos los elementos que se añadan a la lista de visualización por parte de la galería. Este contenedor estará dividido en dos: uno será el lienzo de la imagen ampliada (*lienzo_grande*), y el otro será un lienzo donde estarán contenidas todas las miniaturas (*lienzo_miniaturas*). A su vez, cada figura (grande o miniatura), se compondrá por un marco blanco y la imagen cargada de fichero. La Figura 54 explica de forma visual la estructura comentada, donde cada recuadro corresponde al siguiente atributo:

- A. *contenedor_principal*, de la clase *GaleriaImagenes.as*.
- B. *lienzo_grande*, de la clase *GaleriaImagenes.as*.
- C. *marco_grande*, de la clase *GaleriaImagenes.as*.
- D. *imagen_grande*, de la clase *GaleriaImagenes.as*.

- E. *lienzo_miniatras*, de la clase *GaleriaImágenes.as*.
- F. *marco_minis*, de la clase *ImágenesMini.as*.
- G. *imagen_mini*, de la clase *ImágenesMini.as*.

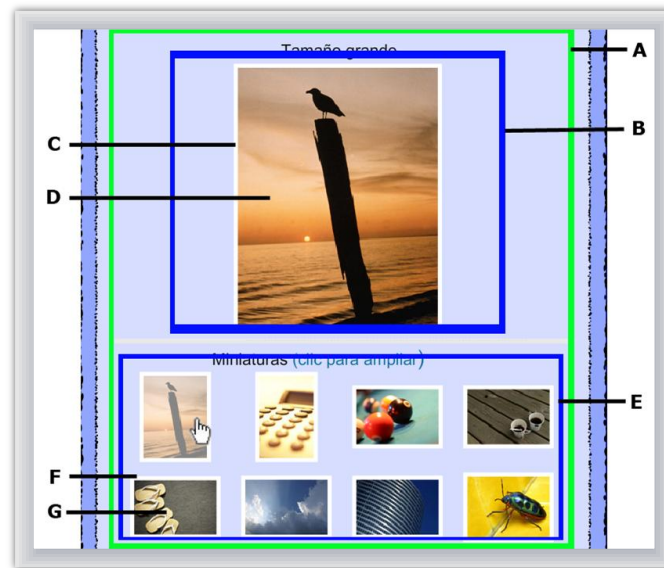


Figura 54. Estructura contenedores de la galería de imágenes.

Las imágenes en miniatura, requieren de un control más complejo, por ser el elemento interactivo y variable de la sección, ya que su número y posición depende de la organización del XML. Por ello se ha agrupado su funcionalidad en la clase *ImágenesMini.as*.

- **Constructor:** Recibe el nombre de la imagen que va a contener, y la ruta de la carpeta donde están todas las miniaturas, y de las grandes, además de una referencia a la clase *GaleriaImágenes*, para poder controlar desde aquí las imágenes originales.

En esta función se inicializan los atributos según los parámetros obtenidos, los contenedores *marco_mini* y *contenedorImagen*, y se coloca inicialmente en (0,0). También se inicializa el atributo booleano *primeraVez* a *false*, y después se llama a *dibujarMini()*.

- **dibujarMini:** Este método carga la imagen a partir de su nombre y la ruta de miniaturas, mediante la clase *flash.display.Loader* (ver 6.2.7 Carga de Archivos Externos). Se añaden los eventos de carga completa y de control de errores.
- **Completado:** Manejador del evento de *Event.COMPLETE* que se lanza al terminar la carga de la imagen. Cumple la función de agregar a *contenedorImagen* el fichero externo leído, y centrarlo dentro de un recuadro de 100x100 px (el tamaño máximo de las miniaturas), en base al tamaño real de la imagen cargada, que se puede obtener únicamente dentro de este evento, a través de su parámetro con:

```
var ancho:Number = e.target.loader.width;
```

```
var alto:Number = e.target.loader.height;
contenedorImagen.x += (100-ancho)*0.5;
contenedorImagen.y += (100-alto)*0.5;
```

Donde *e.target*, hace referencia a *loader.contentLoaderInfo*, que es el elemento que se suscribió al evento y que contiene la información del archivo que se ha leído de disco.

Una vez situada la imagen, se hace la llamada para dibujar el marco.

- **dibujarMarco:** El marco no es más que un rectángulo blanco opaco, que se coloca por detrás de la imagen, de forma que sobresalga 5 píxeles por cada lado. El recuadro se pinta mediante código con la clase *Graphics* (ver 6.2.5 Crear Formas Gráficas desde Código. Uso de la Clase Graphics).
- **añadirMini:** Este es el método que se llama desde *GaleriaImágenes.as*, para añadir a la lista de visualización las imágenes que correspondan, según la página en la que esté el usuario.

Se recibe por parámetro el contenedor, al que se añadirá el marco y la imagen (en ese orden para que el marco quede por debajo), y la fila y columna donde se debe colocar.

Se incluye una animación para que la imagen aparezca en el escenario con una transición de color. La figura comienza transparente, y se va volviendo nítida poco a poco. También se define el *Sprite* contenedor como un botón (ver 6.2.4 Las Clases Movieclip y Sprite), y se añaden los eventos para cuando el ratón pase por encima de la imagen (donde se le asignará una transparencia del 50%), cuando el usuario la pulse (para hacer que aparezca la imagen ampliada, llamando al método *verFotoGrande(ruta)*, de *GaleriaImágenes.as*), y cuando deje de estar seleccionada (volviendo a ser totalmente opaca).

Además de todo esto, a partir de la fila y la columna, se sitúa la fotografía en el escenario, donde *DISTANCIA_ANCHO* y *DISTANCIA_ALTO* son constantes de la clase, y se utilizan para fijar la separación horizontal y vertical entre las figuras con las instrucciones:

```
contenedorImagen.x += (DISTANCIA_ANCHO*col);
contenedorImagen.y += (DISTANCIA_ALTO*fila);
```

Hay una cierta complejidad al ubicarlas en el escenario, ya que existe un problema de concurrencia con la colocación de las imágenes que aparecen nada más cargar, al pasar a esta sección de la interfaz (en este caso, las ocho primeras).

Antes de que termine la carga de una de las primeras imágenes, y se ejecute el método *completado*, se puede haber hecho la llamada para determinar su

posición, según la fila y la columna de la imagen. Debido a que para asignar el valor exacto de la posición, se hace en dos métodos distintos (en *añadirMini* se coloca la fotografía actual respecto al resto de miniaturas, y en *completado*, se modifica ligeramente ese valor, para centrarla en caso de que la figura sea menor de 100x100 px), y estos métodos no tienen un orden fijo de ejecución, cada modificación se debe acumular al valor anterior. Sin embargo, se tiene que diferenciar la primera carga de una imagen, con las posteriores (cuando se cambie de página de muestra en la galería). Aquí entra en juego el atributo *primeraVez*. Al cambiar de página, para que aparezcan las imágenes, se llamará otra vez a la función *añadirMini*, que al asignar la posición, irá aumentando su valor, por utilizar el operador “+=”. Por ello, exceptuando la primera vez que se muestre una fotografía en el escenario, el resto de ocasiones se sobrescribe primero el valor (guardado en atributos), de la variación de la posición que se hace en el método *completado*. Después ya se podrá añadir el desplazamiento necesario por la fila y columna en la que deba estar, sin problemas.

Una vez detallado el proceso que se lleva a cabo, al crear y añadir al escenario cada miniatura, pasaremos a explicar las funciones que se ejecutan a un nivel superior, para manejar el conjunto de la sección, en la clase *GaleriaImágenes*:

- **Constructor:** Al instanciar esta clase, se crea *contenedorPrincipal* y se agrega a una referencia del Main. Se carga el XML con la información sobre las fotografías (ver 6.2.7 Carga de Archivos Externos), y se añaden los eventos propios de control del XML, y de los botones de la navegación interna entre páginas
- **cargarMiniaturas:** Éste es el detector del evento de carga completa del archivo XML. Se obtiene la ruta de la carpeta donde se guardan las miniaturas, y las grandes, y se recorre la lista de nombres de todas las imágenes, creando una instancia de la clase *ImágenesMini*, para cada una de ellas, y guardándola en el *arrayMinis*. Se calcula el número de páginas que habrá, según el número de imágenes que se mostrarán por página (en este caso, ocho, valor guardado en el atributo de esta clase *MAXFOTOSPAG*). Después de tener todas las imágenes, se hace una llamada a *mostrarMiniaturas()*.
- **mostrarMiniaturas:** Esta función se llama al inicio, y cada vez que se cambie de página, para borrar lo que hubiera antes en el escenario, y añadir a la lista de visualización las fotografías que correspondan.

Para ello se ubica *lienzo_miniaturas* en la parte inferior del escenario (la posición que se le da a cada imagen y su marco, se sumará a la posición en

coordenadas del contenedor, que se coloca aquí), y se agrega al *contenedorPrincipal*, junto con *lienzo_grande*.

Una vez hecho esto, se recorre el *arrayMinis*, que es donde se había guardado las instancias de *ImagenesMini*, y se invoca la función *añadirMini*, que añadirá al escenario las figuras con su marco, las recolocará, y le asignará las acciones para verla ampliada. No se recorre la lista completa, sino que se acota el rango, según la página actual, y mientras no se llegue al máximo de elementos para esa página (colocándolos en dos filas de cuatro columnas cada una), o al total de imágenes del XML.

```
var k:Number=(pagina_actual-1)*MAXFOTOSPAG);
while (k <(pagina_actual*MAXFOTOSPAG) && k < total_imagenes)
{...
    arrayMinis[k].añadirMini(lienzo_miniatras, fila, col);
...}
```

- **verFotoGrande:** Esta función se invoca al pulsar sobre una imagen en miniatura, para mostrar la fotografía ampliada en la parte superior de la interfaz. Lo primero que hace, es eliminar *lienzo_grande*, instanciarlo de nuevo, y volverlo a añadir a *contenedorPrincipal*, para borrar lo que pudiera haber antes. Una vez hecho esto, se carga la imagen cuya ruta se recibe por parámetro, con *Loader* (ver 6.2.7 Carga de Archivos Externos), y se añaden los eventos para detectar el momento en el que se complete.
- **cargaGrandeCompletada:** Siguiendo el mismo método que con las miniaturas, al terminar la carga de la imagen, se obtiene su tamaño, y se centra dentro de la zona dedicada a la imagen de máximo 315x315 px. Además de esto, se añade una cantidad para que el resultado ofrezca una figura totalmente centrada en el espacio dedicado a ella del escenario.

```
imagen_grande.x = 155 + Math.round((315-e.target.width)*0.5);
imagen_grande.y = 200 + Math.round((315-e.target.height)*0.5);
```

Finalmente, se dibuja el marco, y se añaden ambos elementos al escenario. El contenedor de la imagen se hace transparente, y se incluye una animación (ver 6.2.6 Crear Animaciones Mediante Código. La Clase Tween), donde que la imagen pasa a ser totalmente nítida y opaca.

- **pagAnterior/pagSiguiente:** Manejadores de los eventos que se reciben al pulsar en los botones *siguiente* o *anterior*. Actualizan el valor de *pagina_actual*, borran el contenido de *lienzo_grande*, y realizan una llamada a *mostrarMiniaturas()*, para que se muestren las imágenes correspondientes a la nueva página.

6.3.6 Cómo Usar/Modificar la Lista de Reproducción de Música

El reproductor de audio de la interfaz flash (ver Figura 26), obtiene la lista de canciones del fichero XML llamado *playlist.xml* de la carpeta *audio*.

Cada archivo de audio está representado por un nodo con la etiqueta “CANCION”, con tres atributos, cada uno de los cuales guarda:

- URL: La ruta de la canción, incluyendo su nombre y extensión.
- TITULO: El título de la canción, para identificarla en la interfaz.
- ARTISTA: El nombre del grupo o cantante, que se muestra como información extra en el reproductor.

```
<PLAYLIST>
  <CANCION URL="audio/mp3/track1.mp3" TITULO="Teenagers"
    ARTISTA="My Chemical Romance" />
  <CANCION URL="audio/mp3/track2.mp3" TITULO="Crank Dat Soulja Boy"
    ARTISTA="Soulja Boy"/>
  ...
</PLAYLIST>
```

En esta sección se añaden las pistas de audio descritas vía XML, a la lista de reproducción de un componente, creado manualmente para el proyecto. Sin embargo, se puede aplicar el conocimiento adquirido al realizar este código, para saber cómo manejar las clases *Sound*, *SoundChannel* y *SoundTransform*, del paquete *flash.media.**, para añadir audio y manipular el volumen del mismo, en cualquier momento y zona de un proyecto Flash que se desee.

También sirve de ejemplo para saber cómo utilizar un componente tipo *fl.controls.List*, cómo cambiar su estilo con *fl.managers.StyleManager* y usar un *DataProvider* (su aspecto gráfico se podrá editar haciendo doble clic sobre la instancia colocada en el escenario, en el software Flash CS4). Asimismo se describe el manejo del componente *fl.controls.Slider*, y cómo aplicar la clase *flash.utils.Timer*, para crear un temporizador que controle el tiempo de duración de las canciones.

Para empezar, comenzamos con una explicación sobre la clase que controla esta sección que es *ReproductorMusica.as*, del paquete *audio*.

- **Constructor:** Se crea una instancia de esta clase, al seleccionar “Música” en el menú principal de la aplicación (ver Figura 14), pasando una referencia del Main por parámetro, para poder acceder a los elementos instanciados en el proyecto Flash. En este punto, se cargan las canciones del XML, y los eventos de fin de lectura de fichero, y de clic sobre los botones de control del reproductor. Se

empieza con el botón de play desactivado, ya que se comenzará a reproducir automáticamente (los botones de play y pausa no están habilitados nunca a la vez, de forma que si se está reproduciendo la canción estará la pausa activada, y el play inhabilitado y con una transparencia del 20%, y viceversa cuando el audio se haya pausado). Utilizando esta propiedad, se bloquea la captura de los eventos del ratón, hasta que se ponga a *true*.

```
m.play_btn.enabled=false;
m.play_btn.alpha=0.2;
```

También se añaden los eventos de control para volver al menú principal, y para mostrar la presentación en caso de querer pausar la aplicación (en ese caso, si hay alguna canción escuchándose, se pondrá en pausa).

Además de todo lo anterior, se incluyen las primeras instrucciones para preparar el temporizador:

- La primera instrucción, inicia el campo de texto de nombre de instancia *tiempo_act*, a cero. Aquí se mostrará la progresión en minutos y segundos de la canción con:

```
m.tiempo_act.text = "00:00";
```

- La siguiente inicializa un temporizador (el parámetro que recibe son milisegundos), y se añade el evento asociado, que hará que se ejecute la función *positionTimerHandler*, cada segundo (en este caso, al pasar en el constructor de *Timer*, 1000, el temporizador se activa cada segundo).

```
positionTimer = new Timer(1000);
positionTimer.addEventListener(
    TimerEvent.TIMER, positionTimerHandler);
```

- **cargarXML:** Esta es la función se ejecuta cuando se completa la carga del XML. Una vez hecho esto, se guardan los elementos en el array *listaCanciones*, de tipo *XMLList*, y se puede crear un estilo para el componente *fl.controls.List* que aparece en la interfaz (ver Figura 26), de la siguiente forma:

- El tamaño de cada fila de una lista, por defecto es reducido. Para cambiarlo, se hace accediendo a la propiedad *rowHeight*, a través la instancia de *List*, que se ha colocado previamente en el escenario.

```
m.listaTemas.rowHeight=35; //nueva altura en píxeles
```

- Para fijar el tipo de letra, color y tamaño, se utiliza *flash.text.TextFormat*

```
var estiloLetra:TextFormat = new TextFormat();
estiloLetra.font = "Arial";
estiloLetra.color = 0x000000;
estiloLetra.size = 20;
```

- Después se indica el nombre del estilo a utilizar, mediante *fl.managers.StyleManager*:

```
StyleManager.setStyle("estiloLetra", estiloLetra);
StyleManager.setComponentStyle(CellRenderer, "textFormat",
                                StyleManager.getStyle("estiloLetra"));
```

- Y se aplica al componente:

```
m.listaTemas.setStyle(
    "textFormat", StyleManager.getStyle("estiloLetra"));
```

Para rellenar la lista, se hace mediante *fl.data.DataProvider* (sirve para los componentes *List*, *DataGrid*, *TileList*, y *ComboBox*). Cada elemento que se agregue a una instancia de esta clase, puede estar compuesto varias etiquetas que guarden distintos datos:

- *iconSource*: Para añadir iconos (*notaMusical* es un símbolo que debe estar contenido en la biblioteca). Además de incluir el nombre en la etiqueta de *DataProvider*, se tiene que especificar para la instancia del componente:

```
m.listaTemas.iconField="iconSource";
```

- *label*: Lo que se incluya aquí aparecerá escrito en la fila correspondiente del componente *List*.
- *data*: Esta etiqueta es información que no es visible de forma externa, pero que también se almacena para cada elemento de la lista. En este caso se pone un valor secuencial para identificar fácilmente cada canción.

```
var dp:DataProvider = new DataProvider();
dp.addItem({ iconSource:notaMusical,
             label:listaCanciones[i].@TITULO,
             data:contador });
```

Para volcar los datos del *DataProvider* en el componente, se asigna a través de la propiedad de la instancia:

```
m.listaTemas.dataProvider=dp;
```

Una vez que se ha colocado toda la información pertinente en el componente *listaTemas*, se suscribe al evento *Event.CHANGE* para detectar cuando se ha seleccionado con el ratón un elemento de la lista (para cambiar de canción).

Además, se activa el componente *fl.controls.Slider*, para que se pueda pinchar y arrastrar, añadiendo los eventos correspondientes para detectar estas acciones (*SliderEvent.CHANGE* al arrastrar, y *SliderEvent.THUMB_RELEASE*, al soltar la barra), y poder avanzar o retroceder proporcionalmente en la canción:

```
m.barraTiempo.liveDragging = true;
```

Por último, se agrega el componente de volumen creado para esta aplicación, instanciando la clase *BarraVolumen* del paquete *com*, y se activa la reproducción de la primera canción de *listaCanciones*.

```
volumen = new BarraVolumen(this);
volumen.x = 480; //se coloca la posición
volumen.y = 305;
m.addChild(volumen); //se agrega al escenario
//Comienza la reproducción del primer elemento
playSong(0);
```

- **playSong:** Esta función recibe como parámetro un número, que indica la posición de *listaCanciones* que se va a reproducir. Un cero indica la posición del primer elemento.

Esta función empieza actualizando el componente *listaTemas* (su elemento seleccionado, y moviendo la barra deslizador en caso de necesidad):

```
m.listaTemas.selectedIndex = num_cancion;
m.listaTemas.scrollToIndex(num_cancion);
```

También modifica los campos de texto para que muestren el nombre y el autor de la nueva canción.

Finalmente carga el fichero audio a reproducir, utilizando la clase *flash.media.Sound*, añadiendo previamente una suscripción al *Event.COMPLETE*, para controlar cuando ha terminado, y abrir el canal de sonido.

```
mySound = new Sound();
mySound.addEventListener(Event.COMPLETE, cancionCargada);
mySound.load(new URLRequest(listaCanciones[num_cancion].@URL));
```

- **cancionCargada:** Este método se ejecuta cuando un archivo de audio ha terminado de cargarse en Flash, y se puede iniciar su reproducción. Para ello, se recoge el objeto, y se activa la música invocando la función *play()*, con la instancia *myChannel*, de *flash.media.SoundChannel*:

```
var soundObject:Sound = e.target as Sound;
myChannel=soundObject.play();
```

Ahora empezará a sonar la canción, por lo que se debe de activar el volumen (para que se oiga) y el temporizador (hasta que no se llame al método *stop*, saltará el evento *TimerEvent.TIMER*, cada segundo, que es lo que se fijó en el constructor).

```
volumen.initVolumen();
positionTimer.start();
```

También se indica para la barra de progreso, el tamaño de la canción, en milisegundos. Esto es necesario, para que cuando progrese el marcador de la barra

a la vez que avanza la canción, el movimiento sea proporcional. Para no salir del tiempo de duración del archivo, se redondea hacia abajo con *Math.floor*.

```
tiempoTotalCancion=soundObject.length/1000;
m.barraTiempo.maximum=Math.floor(tiempoTotalCancion);
```

Luego se pone en el campo de texto *tiempo_total*, la duración de la canción, para que se muestre por una parte la evolución en minutos y segundos, y por otro lado, se dé información al usuario respecto al total. Para darle el formato de String “mm:ss”, se usa la función *tiempoToString* de esta clase, pasando como parámetro los segundos y minutos, en valor numérico.

```
m.tiempo_total.text= tiempoToString(
    Math.floor( (soundObject.length/1000)%60 ),
    Math.floor( (soundObject.length/1000)/60 ) );
```

Para terminar esta función, se incluye la siguiente sentencia, para que al terminar la reproducción de la canción, se pase a la siguiente, de forma automática:

```
myChannel.addEventListener(Event.SOUND_COMPLETE, onNext);
```

- **onNext:** Esta función se encarga de pasar a la siguiente canción de la lista, y si se ha llegado al final, comenzar de nuevo por la primera. Se activa al pulsar en el botón “>”, o cuando una canción ha llegado al final de su reproducción. Se debe parar el canal, y el temporizador, además de reiniciar el contador de tiempo, y la barra de progreso.

```
positionTimer.stop();
myChannel.stop();
resetTiempo();
m.barraTiempo.value=0;
```

Finalmente, llama a *playSong*, con el valor actualizado de *cancion_actual*, y se vuelve a iniciar el temporizador con:

```
positionTimer.start();
```

- **onPrev:** Este manejador de evento se activa al pulsar el botón “<<”. Realiza la función opuesta a *onNext*, volviendo a la canción anterior, y si se ha llegado a la primera, pasar a la última de la lista. El resto de procedimientos respecto a los temporizadores, el canal, la barra etc. son idénticos, ya que hay que siempre se realizan los mismos pasos al terminar una canción y empezar otra.
- **onPause(privado)/pausar(pública):** El primer método se activa al pulsar el botón pausa, y simplemente llama al segundo, que es público, para poder acceder a él desde el Main y parar la música si se ha activado la presentación de Maggie en esta sección. Así no se repite el mismo código innecesariamente.

Al pausar se debe parar el temporizador, desactivar el botón de pausa, y activar el botón de play, y si el canal está activo, guarda la posición de la canción actual antes de parar el canal. Esto último es necesario, para poder continuar después en el mismo punto donde se dejó, y no tener que volver al principio de la canción:

```
posicion = myChannel.position;
myChannel.stop();
cancion_pausa = true;
```

- **onStop:** Función manejadora del evento del botón stop. Se para el canal de audio que se estuviera escuchando, y se reinicia el reproductor a sus valores iniciales.
- **onPlay:** Función manejadora del evento de botón play. Si había una canción en pausa, se comprueba con el atributo booleano *cancion_pausa*, y se continúa la reproducción desde el punto en el que se dejó, desactivando el botón de play, y activando el de pausa.

```
myChannel = mySound.play(posicion);
cancion_pausa=false;
```

Si el reproductor estaba parado totalmente (no está el canal activo), se invoca la llamada a *playSong*, para el valor de *cancion_actual*.

En cualquiera de los casos se activa el temporizador, al ir a comenzar una nueva pista de audio.

El resto de funcionalidades de esta clase, se agrupa en:

- **Tiempo:** Está por un lado, el detector del evento del temporizador que salta cada segundo (*positionTimerHandler*), que llama a *actualizarTiempo()*, para aumentar el contador de tiempo de uno en uno, y ponerlo en formato “mm:ss” con *tiempoToString*. Y por otro lado, *resetTiempo()*, la función que reinicia los contadores de tiempo, y el campo de texto *tiempo_act*.
- **Selección de elemento de la lista:** Cuando se selecciona una canción de la lista, se puede conocer que elemento es, a partir de la etiqueta *data*, donde se había incluido un número secuencial que comenzaba en cero, en el método *cargarXML*. Ahora es donde es de utilidad ese valor, obteniéndolo aquí y pasándolo a *playSong*.

```
cancion_actual=e.target.selectedItem.data
playSong(cancion_actual);
```

Antes de esto, se para el canal con la canción que hubiera antes, se reinicia el tiempo, y la barra de progreso, para dejarlo preparado, y reproducir la pista seleccionada.

- Volumen: El método *setVolumen*, se invoca al pulsar sobre el icono del volumen, desde la clase de ese componente *BarraVolumen*, para actualizar el nivel de audio del canal, en función de donde ha pulsado el usuario, pasando como parámetro el valor numérico al que debe pasar el volumen. Esto se hace mediante la clase *flash.media.SoundTransform*, y la propiedad correspondiente del canal, de la siguiente forma:

```
var soundTrans:SoundTransform = new SoundTransform();
soundTrans.volume=volumenNuevo;
myChannel.soundTransform=soundTrans;
```

- Control de la barra deslizador:
 - *barraTiempoModificada*: Se activa al pulsar sobre el componente Slider, y arrastrar. La música se irá actualizando, así como el contador de tiempo, a la vez que se mueve la barra, por lo que sonarán trozos de audio mientras tanto, si la pista estaba reproduciéndose.
 - *sliderLiberado*: Se activa al soltar la barra, momento en el que se actualiza la posición del canal, según lo que marque el Slider. Ese valor se obtiene a partir del objeto destino del evento mediante la propiedad *value*.

```
posicion=e.target.value*1000;
```

Después, se continúa con la reproducción y se actualizan los contadores de tiempo.

Una parte importante, que se ha comentado, pero no detallado, durante la explicación del código del archivo *ReproductorMusica.as*, es lo referente al volumen. Dado que el reproductor de la aplicación es de manufactura casera, porque en la versión de Flash CS4 este componente no se incluye completo como en la anterior, se decidió hacerlo desde cero, gráfica y funcionalmente. La clase que desempeña la labor del control del volumen es *BarraVolumen*.

El símbolo gráfico al que está asociada, consta de los siguientes elementos:

- volume_mc.backBar_mc*. Barra que engloba el desplegable del volumen. Oculta
- volume_mc.bar_mc*. Barra que marca el nivel máximo de volumen. Altura: 1px.
- volume_mc.indicator_mc*. Barra que marca el nivel de volumen. Altura: 1px.
- volume_mc.hitMe_mc*. Zona activa invisible donde se reconoce el clic de ratón.
- volume_mc.vol_btn*. Botón que despliega la barra. Al pulsar, silencia el audio.

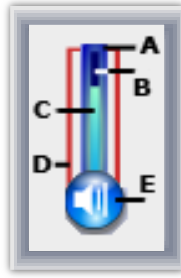


Figura 55. Estructura del componente de volumen.

El nombre de la instancia del símbolo completo es *volume_mc*. Los puntos anteriores son elementos dentro de él, por lo que para acceder a ellos en el código, se hace con el operador “.”.

Algunos elementos están ocultos, y otros se han reducido a una altura de un píxel, para que la barra no sea visible en un inicio, a menos que se pase el ratón por encima del botón. La animación donde se despliega la barra, se incluye en el código, aumentando la altura o desplazando los símbolos (ver 6.2.6 Crear Animaciones Mediante Código. La Clase Tween).

A continuación se incluye un breve resumen del funcionamiento de la clase, donde se hará referencia a las zonas descritas de la Figura 55.

- **Constructor:** Al crear una instancia de esta clase, se obtiene una referencia por parámetro de tipo *ReproductorMusica*, para poder cambiar el volumen del canal, cuando el usuario pulse sobre el componente del volumen. En el constructor se inhabilita la zona D, para que no cambie el icono del ratón, detectando que es una zona de tipo “botón”, hasta que no se despliegue el menú, al pasar por la zona E.

Además, se fija el volumen inicial por defecto al 75% (el rango de valores que se admiten es [0,1]), y se inicia a *false* los atributos *volumeOFF* (para controlar si se ha silenciado), y *volumeExpanded* (para controlar cuando se ha desplegado la barra).

- **initVolumen:** Esta función se llama desde la clase *ReproductorMusica* para darle el valor inicial al volumen, y que se escuchen las pistas de audio, y añadir los eventos para desplegar la barra (*MouseEvent.ROLL_OVER*, cuando el ratón pasa por encima de la zona E), cerrarla (*MouseEvent.ROLL_OUT*, cuando el ratón sale de la zona E), y silenciar el volumen (*MouseEvent.CLICK*, en la zona E).
- **silenciar:** Alterna entre el nivel estándar de volumen del componente, y el silencio, con cada clic de ratón. Varía el tamaño de la zona C, con una animación y de forma proporcional al volumen, haciéndola ocupar el 75% de la barra, o desapareciendo por completo cuando se quite el volumen.

- **desplegarBarraVolumen:** Despliega las zonas A, B y C, con una animación, además de activar la zona D, cuando se ejecuta este detector de evento. Añade un evento para responder ante los clics de ratón que se produzcan en la zona D, para cambiar el volumen según la posición del ratón cuando pulsó sobre esta zona.
- **cerrarBarraVolumen:** Realiza las animaciones inversas a las que se hacen en *desplegarBarraVolumen*, para hacer desaparecer las zonas A, B y C, además de desactivar la zona D.
- **updateVolumen:** Recoge la coordenada donde se encuentra el puntero del ratón sobre la zona D, para calcular el nuevo valor del volumen, y pasarlo a *ReproductorMusica*.

```
volumenActual = Number(volume_mc.hitMe_mc.mouseY / -100);
```

Además añade una animación para que la zona B se acople al valor nuevo que se le haya dado al volumen.

6.3.7 Cómo Usar/Modificar la Reproducción de Videos

El reproductor de videos (ver Figura 27), obtiene la lista de películas a visualizar a partir de un archivo XML, con la siguiente estructura:


```
<PLAYLIST loop ="si" skinVideo="video/SkinUnderPlayStopSeekMuteVol.swf">
  <VIDEO URL="video/films/RatatouilleOptimo.flv" TITULO="Ratatouille"/>
</PLAYLIST>
```

De esta forma se pueden cambiar los videos que se reproducirán en la interfaz, sin tener que ir al código fuente, y volver a compilar el proyecto Flash.

Cada video irá en un nodo de etiqueta “VIDEO”, y su ruta se deberá especificar en el atributo “URL”. Se ha creado una carpeta llamada *films*, dentro del paquete *video*, donde guardar todas las películas que se quieran mostrar. Es importante tener en cuenta, que los videos que se vayan a reproducir en una aplicación Flash, deben de tener la extensión *.**flv**. Los videos habituales *.avi, *.mpg, *.wmv, o similares no funcionan, por lo que se debe utilizar un programa que los transforme al formato correcto. La interfaz actual incluye la película de Ratatouille como ejemplo, convertida a partir de un archivo *.divx, con una versión de prueba del programa de AVS Video Converter¹³, por lo que aparece una marca de agua intermitente durante la reproducción. Este software permite una alta calidad en su modo de compresión óptima, pudiendo ampliar la imagen de la película, para que se ajuste al tamaño de la interfaz, sin perder calidad.

En cuanto al resto de etiquetas, “loop” se utiliza para saber si la reproducción de películas se hace de forma cíclica, o si debe de parar cuando se hayan visto todos los videos de la lista. En este caso, ya que sólo hay un video en la lista, este se reproduce de forma continua.

El aspecto del componente de video, también se puede variar, modificando la ruta del atributo “skinVideo”. De momento se escogió *SkinUnderPlayStopSeekMuteVol.swf*, añadiéndolo a la carpeta perteneciente al paquete *video*.

Los distintos archivos *.swf que definen el aspecto del componente de video, aparecen en la carpeta donde se encuentre el archivo *.fla, al publicar o compilar un proyecto con un FLVPlayback en el escenario. Según el valor que se haya asignado a la propiedad *skin* de este elemento, así Flash agregará el archivo correspondiente. Se puede ver una vista previa de todos los formatos disponibles en el inspector de componentes, al que se accede pulsando al botón  que aparece en el inspector de propiedades, cuando está seleccionado el elemento en el escenario. La siguiente figura muestra como son las ventanas, y como acceder a la vista previa:

¹³ AVS Video Converter - <http://www.avs4you.com/AVS-Video-Converter.aspx>

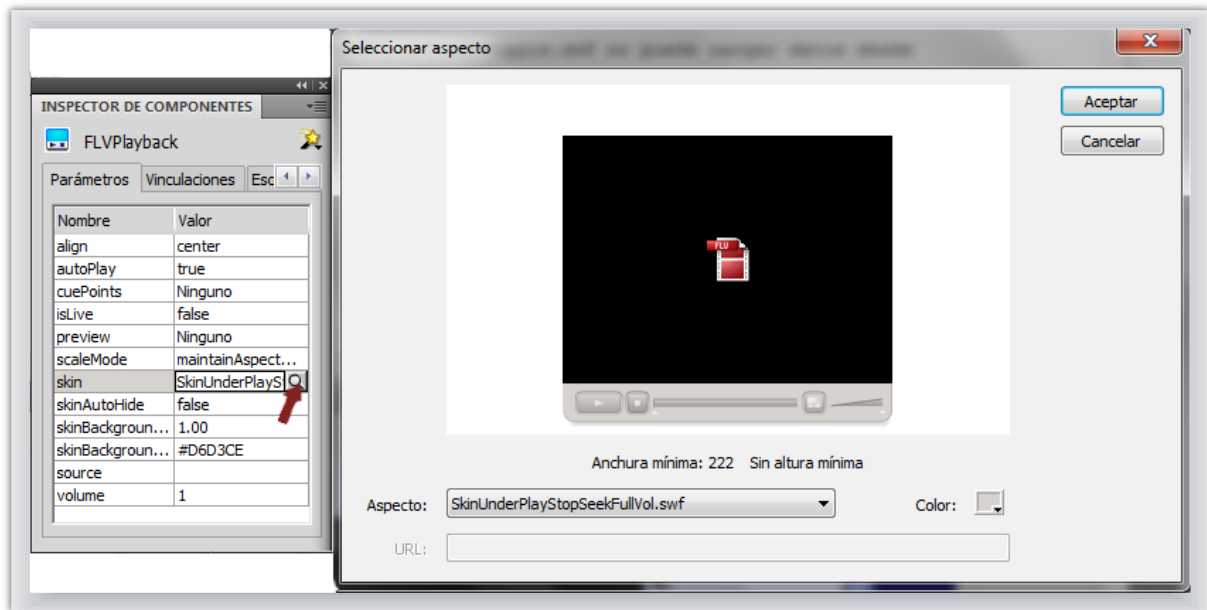


Figura 56. Inspector de componentes y vista previa de FLVPlayback.

Una vez obtenido el archivo, se podrá mover al paquete de video y se deberá actualizar el nuevo nombre y la ruta en el XML, para que se aplique el cambio en la siguiente ejecución.

Es mejor utilizar un formato que no tenga botón de maximizar, porque si un video se pone a pantalla completa, habrá parte de la imagen que no se vea, debido al tamaño reducido del hueco en el pecho de Maggie.

Este apartado en un principio podría resultar muy sencillo, debido a que Flash provee un componente de video bastante completo, y con propiedades fácilmente configurables. Sin embargo, en la práctica, se descubrió que aunque en las compilaciones rápidas desde Flash funciona todo bien, al publicar a *.exe, y ponerlo a pantalla completa, la reproducción de video daba bastantes problemas: no se escuchaba, o aunque se pudiera oír el audio, sólo había una pantalla negra que cubría toda la ventana, lo que imposibilitaba la visualización.

Por todo ello, se decidió crear y controlar el componente mediante código en una clase *Video.as*, y fijar todas las propiedades a los valores deseados, manualmente. De esta forma, se resolvía el problema cuando se ejecutaba la publicación Flash. A causa de incidentes como éste, es por lo que se sugiere no probar los archivos Flash sólo con la compilación rápida, sino probar también el funcionamiento del software al ejecutar la versión publicada, ya que el comportamiento puede variar bastante, encontrando errores en zonas de código que ya se creían plenamente funcionales.

Para dar una idea como se implementó esta clase, se presenta un breve repaso al código de la misma:

- **Constructor:** Se inicializa el atributo *cont_video*, de tipo *FLVPlayback*, definiendo su tamaño y posición (para ajustarlo a la ventana, y que se vea lo más grande posible), además de las siguientes propiedades:

- La propiedad *fullScreenTakeOver* debe ser *false*, para evitar que el objeto de visualización ocupe toda la pantalla al maximizar la ventana, tapando el resto de elementos.

```
cont_video.fullScreenTakeOver= false;
```

- *Autoplay* con valor *true* implica que no es necesario hacer una llamada a *play()* al crear un *FLVPlayback*, para hacer que reproduzca un video. Comienza automáticamente en el momento que se asigna una dirección a la propiedad *source*.

```
cont_video.autoPlay=true;
```

- Para que el video mantenga un tamaño proporcional de alto y ancho, según el tamaño asignado al *FLVPlayback*, y se centre en el escenario, se tienen que incluir las instrucciones:

```
cont_video.scaleMode=VideoScaleMode.MAINTAIN_ASPECT_RATIO;
```

```
cont_video.align = VideoAlign.CENTER;
```

Una vez bien definido el componente, se lee el archivo XML *playlist_video.xml*, que se encuentra dentro de la carpeta *video*, y se añaden los detectores de eventos de control de la navegación y para pausar la aplicación.

- **Cargaxml:** Este método se ejecuta cuando ha finalizado la carga del XML. En él se asigna la propiedad *skin*, el color del componente *FLVPlayback*, y la ruta y el volumen del video a reproducir. Después se añade al escenario de la forma habitual, y se agrega un evento *VideoEvent.COMPLETE*, para controlar cuando ha finalizado:

```
cont_video.skin = myXML.@skinVideo;
```

```
cont_video.skinBackgroundColor = 0x999999; //color gris
```

```
cont_video.source = xmlList[0].@URL; //se empieza por el primero
```

```
cont_video.volume = 1; //volumen al máximo
```

```
m.addChild(cont_video);
```

- **videoFinalizado:** Manejador del evento ante el fin de la reproducción de una película. Buscar en la lista si hay más películas, para continuar con la siguiente, o en caso de ser la última, según el valor obtenido de la etiqueta "loop" del XML, se continuará por el primero, o se parará.

Además de estos métodos, se incluyen los manejadores para volver al menú principal, y pausar la interfaz. En el primer caso, se eliminará el componente de video. En el segundo, se pausará, y al salir de la presentación, será el usuario quien lo tendrá que activar de nuevo.

6.3.8 *Cómo Hacer otra Versión del Sapientino*

En este punto se explicará cómo se hizo el juego Sapientino de la aplicación Flash, y los detalles básicos a tener en cuenta para poder realizar una versión del mismo, con cualquier otra temática de fondo diferente.

El objetivo de todo juego denominado Sapientino, es simple: unir conceptos de forma correcta. Estos pueden representarse como una lista de preguntas y respuestas; palabras y definiciones de las mismas; locuciones en un idioma, y su traducción; un mapa y nombres de capitales, provincias o accidentes geográficos; nombres de animales e imágenes con su representación gráfica... Tantas variantes como se imaginen, diseñadas de múltiples formas (con listas, botones, gráficos, animaciones, sonidos...).

En el caso actual de este proyecto, la idea fue crear un juego con un componente más visual y atractivo, por lo que se quiso incluir una imagen que ocupara gran parte de la pantalla, y por otro lado, botones, que pudieran seleccionarse fácilmente pulsando con los dedos en una pantalla táctil. De esta forma, se crearía a partir de una fotografía estática, una cierta animación de los elementos implicados en el juego, manipulándolos previamente con Photoshop¹⁴. Y por otra parte, se diseñarían unos botones donde aparecerían los nombres de aquellos elementos interactivos de la imagen (ver Figura 21).

Como añadidura a los componentes básicos, se quiso incluir la posibilidad de pedir una pista, y que Maggie fuera la encargada de decirla con su voz, en inglés o español, según la versión del programa que se ejecute.

Hubo varios motivos tras esta decisión. Por un lado, se quería dotar de mayor interactividad al juego, y también, que tuviera un cierto componente educativo, ya que las pistas se basan en dar características biológicas, y de conducta de los animales y plantas de una laguna, para ayudar a identificarlos. También se pretendía dar un mayor uso de las habilidades disponibles de Maggie, como la voz, que es una de las áreas más activas, donde está trabajando actualmente el personal del laboratorio. Por ello era conveniente incluir un caso especial para los mensajes de voz en el protocolo de comunicación (ver Tabla 30 RNF-13. Protocolo de mensajes), al analizar que era una habilidad a la que se podría recurrir fácilmente, y con mayor frecuencia. Igualmente, todo esto servía para acentuar la utilidad de la conexión por socket, y mostrar una parte del potencial de uso que puede tener la aplicación.

¹⁴ Adobe Photoshop - <http://www.adobe.com/es/products/photoshop/photoshop/>

Una vez explicado en líneas generales, se ha de puntualizar las distintas fases de trabajo a tener en cuenta, para realizar este juego, aplicable a otros a los que se les quiera dar una estructura similar.

Para empezar, una vez elegido el tema y el formato, si se va a utilizar una fotografía estática, y modificarla como se hizo en este Sapientino, esto se debe de realizar preferentemente en un programa de edición de imágenes. También se podría realizar desde Flash CS4, pero para aquellos que tengan algún conocimiento en Photoshop, o con software de características similares, esta parte será más sencilla de realizar, siguiendo estos pasos por separado del proyecto Flash:

- Con la fotografía cargada en el programa, y teniendo en cuenta que los elementos se deben de poder seleccionar con el dedo (tienen que tener una cierta área mínima), se recortan, una por una, todas las zonas que se van a utilizar en el juego. El resultado será mejor si se delinea bien el contorno de la figura, para evitar más tarde que se mezclen varias figuras que estén juntas. Por ejemplo:

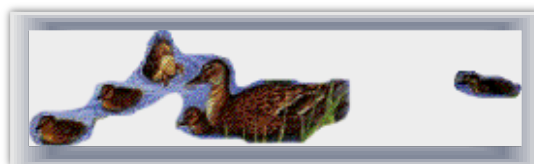


Figura 57. Ejemplo de recorte utilizado en el juego *Sapientino*.

- Es muy importante que cada una se guarde en un archivo por separado, con un fondo transparente. Esto es necesario para que al colocarlo en Flash, no aparezcan partes blancas rodeando la imagen, que revelen el recorte realizado.
- El formato de las figuras debe ser *.png, o *.jpg, para que se pueda exportar a la biblioteca del proyecto Flash en el que se esté trabajando.
- Una vez dividida la imagen, se debe de aplicar un filtro a cada parte recortada, que las aclare y dé luminosidad, y guardar la nueva figura por separado. Esto servirá para crear después el efecto de iluminación en las zonas seleccionables de la fotografía, cuando se pase el ratón por encima de ellas. Ejemplo:



Figura 58. Ejemplo de recorte con filtro de luz, utilizado en el juego *Sapientino*.

Cuando se tienen las imágenes de cada elemento interactivo, en su modo normal, y la copia “iluminada”, y están exportadas dentro de la biblioteca, se puede pasar a crear la estructura gráfica del juego en Flash. Para esto, se dieron los siguientes pasos:

- Se crea un símbolo de botón, para cada recorte realizado (ver 6.1.5 Crear Botones). En el fotograma “Reposo”, se coloca la imagen normal, y en los fotogramas “Sobre” y “Presionado”, la figura con el filtro. De esta forma, se crea el efecto de iluminación al activar el botón, y así, diferenciar en el juego las zonas interactivas, de las estáticas. Se aconseja colocarlos en las coordenadas (0,0), para que la posición entre fotogramas sea la correcta.
- En el fotograma “Zona Activa”, lo más sencillo es copiar otra vez cualquiera de las imágenes de los fotogramas anteriores. Sin embargo, esto hará que el rectángulo que contenga la figura, y no sólo la forma que se ha cortado, interaccione con el ratón, lo cual puede dar resultados no deseados, si hay figuras muy juntas, y sus zonas activas se solapan. Por esto, es conveniente dibujar el contorno de los recortes a mano en Flash, y después, rellenar el interior, para que sólo la zona que corresponda exactamente a la forma de la imagen, responda al paso del puntero del ratón.
- Una vez terminados todos los botones, se debe colocar la fotografía completa original en el escenario, y poner una instancia de cada uno de los botones de las imágenes, sobre su posición correspondiente en el conjunto, de forma que queden totalmente disimulados. Para que quede bien, no se debe de haber modificado el tamaño en ningún paso del proceso.
- Para cada instancia, se debe de asignar un nombre representativo, al que se pueda llamar desde el código, más tarde, a la hora de programar el control del juego.
- Cuando ya se tiene la parte de la fotografía preparada, se deben de añadir los botones con los nombres de cada elemento interactivo, colocándolos por el espacio libre del escenario. En este caso, son los nombres de los animales, insectos o plantas que se tienen que averiguar durante el juego.
- Cualquier otro componente que se contemple en el diseño (como el botón de pistas, el botón para volver atrás, y el logo del laboratorio para poner en pausa la interfaz y mostrar la presentación), se incluye en el escenario y se les da a todos ellos un nombre identificativo de instancia, para referenciarlos desde ActionScript.
- Por último, se aconseja organizar la biblioteca en carpetas, ya que este juego implica un número elevado de símbolos, con los distintos botones e imágenes.

Llegados a este punto, ya se habrá creado toda la parte gráfica del juego, y faltará programar la funcionalidad adecuada (explicada en el punto 4.5.2 Juego Sapientino), para conectar las respuestas de los distintos eventos que se produzcan, al pulsar los botones que hay en el juego.

En este proyecto, la clase que recoge estas instrucciones es *Sapientino.as* del paquete *sapientino*. Dentro de este directorio, se encuentra el archivo *pistas_sapientino.xml*, que como su nombre indica, contiene las frases que se enviarán a Maggie, para que narre las pistas.

Primero hay que explicar el contenido del XML. El archivo está estructurado por nodos con la etiqueta “PISTA”, con dos atributos “ID” y “FRASE”. El primer atributo sirve para saber a quién corresponde, la pista que aparece en el texto del segundo atributo. Es importante controlar el tamaño de esta frase, para que unido a la cabecera del mensaje, no se sobrepasen el máximo de cien caracteres, ya que en ese caso, no se procesará correctamente en el servidor, y Maggie permanecerá en silencio. En el nodo principal “LISTA”, se incluye el atributo “intro”, que será el mensaje que se enviará a Maggie para que haga un comentario sobre el juego, en su inicio.

```
<LISTA intro = "En este juego debes identificar animales, insectos o
plantas típicos de un entorno acuático." >
  <PISTA ID = "rana"
    FRASE = "Anfibio de patas palmeadas. Alternan periodos de
vida acuática y terrestre."/>
  <PISTA ID = "sapo"
    FRASE = "Anfibio de cuerpo rechoncho, ojos saltones y
piel granulosa."/>
  ...
</LISTA>
```

El nombre que aparece en “ID”, principalmente sirve para no confundirse a la hora de escribir las pistas, porque en el código, se cogerá una frase u otra en base al orden en el que están puestas en el XML. En la clase *Sapientino.as*, se han incluido nueve constantes, para hacer referencia a esta disposición, que se debe de respetar rigurosamente:

```
private var RANA:Number = 0;
private var SAPO:Number = 1;
private var CABALLITO:Number = 2;
private var NENUFAR:Number = 3;
private var PATO:Number = 4;
private var ZAPATERO:Number = 5;
private var ESPADAÑA:Number = 6;
private var CULEBRA:Number = 7;
private var RENACUAJO:Number = 8;
```

Condicionar el XML de esta forma, servirá para realizar búsquedas directas a la posición exacta, evitando introducir más bucles en el código, y sentencias de comparación, ya que habría que crear casos específicos para cada elemento. De esta forma se ha tratado de programar de la forma más sencilla posible, de forma que si se quiere aplicar a juegos más extensos, no implique modificar y añadir un número elevado de líneas de código.

En cuanto al resto de atributos necesarios, para gestionar y almacenar el estado del juego, son:

- *id_seleccionado*: Contiene el valor de la constante que corresponda, según el botón de la lista de nombres que se pulse en la interfaz de la Figura 23.
- *array_acertados*: Lista que contiene unos y ceros, según si el elemento de esa posición se ha acertado ya o no. Sirve para controlar cuando se ha terminado, y para que los botones de una pareja unida correctamente, queden desactivados el resto del juego. Se inicializa al valor de la constante *NUM_ELEMENTOS*, que será el número de elementos a adivinar en el juego.
- *pistas_activadas*: Variable booleana que controla si se han pedido pistas, para que Maggie proponga intentarlo de nuevo, pero esta vez, sin ayuda.
- *fallos*: Almacena el número de veces que se ha fallado, para calcular las puntuaciones.
- *xmlList*: Lista donde se guardarán las pistas obtenidas a partir del XML.
- *m*: Referencia a la clase Main.

En cuanto a la programación, debido al número de botones, la mayor parte del código se centra en las respuestas de los eventos de *MouseEvent.CLICK*.

Para dar inicio a este juego, se hace en el Main, en el método *controlNavegación*, al recibir el evento que se produce al seleccionar “Sapientino”, en el menú de juegos. Se pasa al fotograma que corresponde a la interfaz izquierda, de la Figura 21, y se instancia la clase, pasando como parámetro una referencia al Main, con la palabra reservada *this*, para de esta forma tener acceso a los símbolos instanciados en el escenario:

```
var juegoLaguna:Sapientino = new Sapientino(this);
```

Además de esto, se añaden los eventos de control para volver al menú de juegos, y para mostrar la presentación de Maggie.

Una vez en este punto, se queda a la espera de que el usuario pulse el botón de inicio para comenzar, pasando al siguiente fotograma donde estará la estructura gráfica diseñada, con los botones y la imagen de la laguna. También se dará la instrucción para cargar el XML, se invocará

el método *desactivarImagenes*, y se asignará la misma función manejadora, *botonSeleccionado*, para todos los botones de la lista de nombres.

La línea general, una vez hecho todo lo anterior, se activará con la interacción del usuario. Si no existe esa interacción, no ocurrirá nada.

1. En un inicio, sólo estará la lista de botones activados, por lo que si el usuario pulsa uno, se ejecutará *botonSeleccionado*, donde se realizarán los siguientes pasos:

- a. La selección se almacenará en la variable *id_seleccionado*, identificando el objeto destino del evento, mediante las propiedades *target.name*, y comparándolo con los nombres de instancia de los botones.
- b. Las figuras interactivas camufladas en la fotografía, se activan, pasando a desactivar los botones con los nombres:

```
activarImagenes();
desactivarBotones();
```

Esto se hace para que una vez seleccionado un elemento, se tenga que probar a dar una respuesta, antes de pasar a seleccionar otro nombre.

- c. Finalmente, se activa el botón *btn_pista*, que sólo estará disponible cuando falte por distinguir la imagen que corresponde al nombre escogido.

2. Después, el usuario podrá optar por dos caminos:

- a. Si tiene dudas respecto a que imagen seleccionar, puede pulsar el botón para pedir una pista (siempre será la misma, pero se podrá repetir si no se ha entendido bien, pulsando de nuevo). Esto activa el método *peticionPista*, donde:
 - En caso de ser la primera pista se marcará como verdadero la variable booleana *pistas_activadas*.
 - Se obtendrá de *xmlList*, según el valor de *id_seleccionado* la frase de la pista que toque, y se enviará un mensaje a Maggie, añadiendo la cabecera adecuada para datos de voz.
- b. Si se pulsa directamente en uno de los botones con las imágenes de la fotografía, se ejecutará el método *imagenSeleccionada*, que comparará si se ha acertado o no.
 - Si es correcto, se marca con un uno en la posición dada por *id_seleccionado*, de *array_acertados*, y se elimina la suscripción al evento de la pareja de botones nombre/figura, además de desactivarlos para que no se puedan volver a seleccionar, y se envía un mensaje, para que Maggie felicite por el acierto.

- Si era incorrecto, se aumenta en uno la variable *fallos*, y se envía un mensaje para que Maggie de aviso del error cometido.

Además, se comprobará dentro de esta función, si se ha llegado al final del juego, después de acertar la última pareja que quedaba (esto es, cuando todas las posiciones de *array_acertados* tienen un uno, lo que se prueba en *comprobarFinJuego*, que devolverá un valor booleano con el resultado).

- En caso afirmativo, se pasará al fotograma con la interfaz de la Figura 24, para mostrar las puntuaciones (método *calcularPuntuacion*), y se añadirán las suscripciones de los botones que permitirán volver al menú de juegos, o mostrar la presentación.
- En caso negativo, se volverá a desactivar las imágenes y activar los botones con los métodos:

```
desactivarImagenes();
activarBotones();
```

Y el juego quedaría a la espera de nuevas interacciones por parte del usuario, con lo que se volvería al punto 1, de forma que en cada ronda se irán eliminando posibilidades, para que no se juegue con el mismo elemento varias veces.

Con esta explicación, ya se conoce el proceso general del Sapiéntino, a falta de explicar un par de puntos más como son:

- Las puntuaciones:
 - Se calculan en porcentaje, a partir de la variable *fallos*, y contando con los nueve aciertos que siempre se han de producir para terminar el juego.
 - Se muestran los resultados en los campos de texto del fotograma 14, de nombre de instancia, *txtFallos* y *txtAciertos*.
 - Y se envía un mensaje a Maggie, para que sea ella quien comente el resultado final del juego.
- Desactivar/activar los botones y las imágenes: Estos métodos son muy parecidos, y siguen el mismo proceso para los dos casos. La única diferencia radica en que los botones de la lista de nombres, se les da una cierta transparencia, para diferenciar aquellos que están desactivados, y al activar vuelven a ser opacos.

En ambos casos, se comprueba para todas las instancias, si corresponde a un par de elementos que aun no se haya emparejado, y si es así:

- Para añadir y habilitar el botón se hará:

```
m.img_rana.addEventListener(
    MouseEvent.CLICK, imagenSeleccionada);
m.img_rana.enabled = true;
```

- Y para deshabilitar se hará:

```
m.img_rana.removeEventListener(
    MouseEvent.CLICK, imagenSeleccionada);
m.img_rana.enabled = false;
```

El código que se utiliza en esta clase es muy sencillo, y si se ha comprendido como se ha utilizado el resto de apartados del capítulo 6.3 Manual de la Aplicación, así como todos los ejemplos y procesos descritos en los apartados 6.1 Breve Manual Introductorio a Flash y, 6.2 Breve Manual Introductorio a ActionScript 3.0, será fácil seguir las sentencias utilizadas en la clase *Sapientino.as*. Principalmente, se deben realizar las mismas acciones para las instancias del mismo tipo de símbolos, por lo que las instrucciones son breves, aunque repetitivas. Por ello se han tratado de agrupar en métodos, para que resulte más legible.

Para añadir juegos nuevos, se deberá incluir el botón correspondiente en el menú de juegos, y añadir en la clase Main, dentro del método *controlNavegación*, como un bloque *if* más el código de control para cargar el juego, y mover al punto de la línea de tiempo donde estén los fotogramas que hagan de fondo.

Si el juego está en un archivo flash *.swf por separado, se podrá añadir desde código, en el momento en el que se seleccione, liberando parte del peso del proyecto principal. Sin embargo, en ese caso, se tendrá la desventaja de que no se podrán enviar ni recibir mensajes, ya que se trata de archivos Flash compilados, con su propia línea de tiempo y código, independientes de la aplicación sobre la que se ejecuten.

7 Planificación

En este capítulo se especifica el tiempo de duración de las distintas tareas o actividades realizadas a lo largo del proyecto.

La herramienta elegida para exponer la planificación es el Diagrama de Gantt, donde se representará, gráficamente, la duración de cada etapa a partir de la fecha de comienzo y de fin, utilizando de base un calendario.

A continuación, en la Figura 59 se muestra el diagrama de Gantt, donde aparece el presente proyecto, dividido en las diversas tareas que se han llevado a cabo, para completar este trabajo.

Esta representación, es la planificación real que se ha efectuado. Se observa un período de menor productividad, debido a que fue necesario compaginar dos trabajos, junto con el desarrollo del proyecto, unido al tiempo en el que Maggie necesitó una serie de reparaciones, antes de poder volver a un nivel de labor adecuada. Debido a ese retraso, a partir de mayo, la actividad se multiplicó, para recuperar el tiempo perdido.

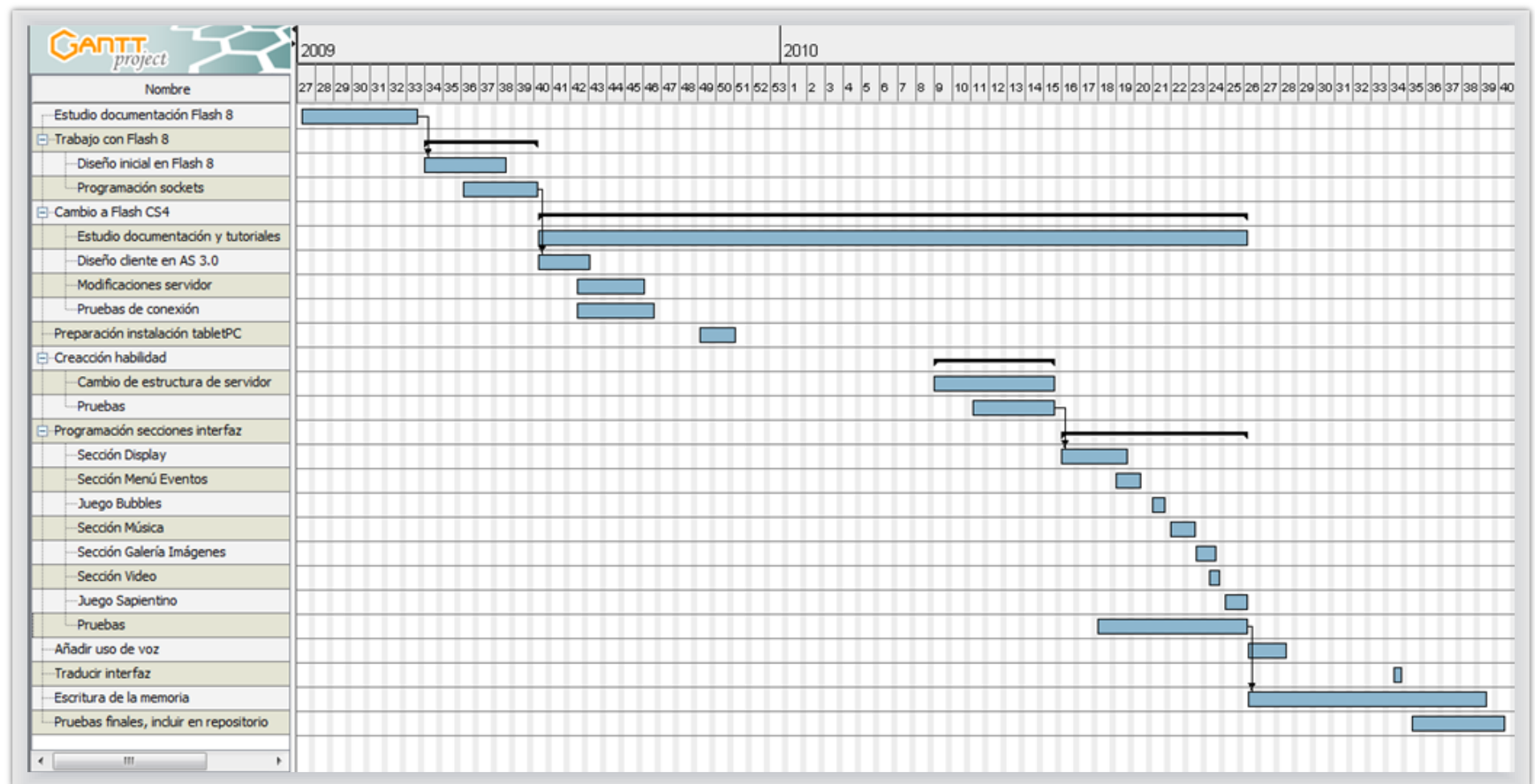


Figura 59. Diagrama de Gantt.

8 Trabajos Futuros

Como ya se ha comentado a lo largo de esta memoria, el trabajo aquí presente es el inicio de lo que se espera sea un medio ampliamente utilizado, por el personal investigador del departamento de robótica, de la Universidad Carlos III de Madrid.

Por ello se hizo necesario incluir el capítulo 6 Manuales. Por ello se ofrecen unas pautas a seguir que faciliten el uso, y las futuras alteraciones, de cualquier parte de la interfaz diseñada.

De esta forma, los trabajos futuros más inmediatos relacionados con este proyecto, son:

- Añadir juegos y secciones, hechas en Flash, en los que ya están trabajando algunas personas del laboratorio.
- Comenzar a darle un uso real a la sección detallada en el punto 4.4 Display de Estados, una vez que el investigador que participó en su diseño, vuelva de su estancia en el extranjero en los próximos meses.
- Modificar la presentación actual de Maggie, que aparece al pulsar el logo del laboratorio de robótica. Se podría añadir mensajes por voz, según avanza la animación, donde Maggie se presentase a sí misma.

Otra idea que surgió, fue la de crear nuevas versiones del Sapientino, donde se pudiera seleccionar por voz las parejas de elementos; o bien, contestar en voz alta a preguntas que hiciera Maggie; o simplemente, cambiar los gráficos manteniendo el formato actual, pero en vez de una laguna, por ejemplo, que fuera un mapa y donde situar provincias o capitales. Son muchísimas las variantes que se podrían realizar a partir de este estilo de juegos.

También se podría asociar una base de datos más elaborada que un simple XML, donde aplicar parte de la inteligencia artificial del robot, y crear un almacenamiento externo similar a la memoria a largo plazo de Maggie.

Se podría guardar, por ejemplo, fotografías de las diversas personas que habitualmente trabajan en el departamento, y que cuando se activara la habilidad de reconocimiento facial de Maggie, al reconocer a una de ellas, apareciera en Flash la foto de esa persona. Incluso, modificando la comunicación por sockets para que se pudiera realizar una comunicación FTP, se podría enviar archivos completos, y las imágenes almacenadas podrían ser capturas de la propia cámara de Maggie.

Otra opción sería que una vez que la habilidad de reconocimiento del habla esté más evolucionada, que cuando la persona que esté hablando con Maggie mencione ciertos temas, desde la interfaz Flash, se muestre una galería de imágenes, con fotografías seleccionadas de la

base de datos, que tuvieran una cierta relación con el tema de conversación. O en más bajo nivel, relacionadas con palabras clave que dijera un interlocutor, y que captase Maggie.

En general, se espera que con el tiempo se conecte la aplicación Flash con un mayor número de las habilidades disponibles en Maggie. No solo con la voz, o la cámara, o controlar el movimiento de la base, o de los brazos, sino a un más alto nivel. Una idea podría ser trabajar con el planificador, para indicar a través de la interfaz Flash un cierto recorrido. O para informar sobre un plano, los movimientos a realizar para llegar a una posición determinada, y que pudiera conectarse con la parte reactiva de los sensores de colisión, para salvar los obstáculos no incluidos en ese plano inicial.

Las posibilidades, son ciertamente muy amplias.

9 Conclusiones

Desde un comienzo, este proyecto se planteó como un reto, debido a varios requisitos con los que se debía trabajar, como eran:

- Utilizar un programa de desarrollo como es Flash, y su correspondiente lenguaje de programación, ActionScript, con los que no estaba familiarizada, y que debía aprender desde cero y emplear para realizar la interfaz gráfica. Esto se agravó, cuando se hizo patente la necesidad de modernizar las aplicaciones realizadas en Flash hasta la fecha, a sus últimas versiones, lo que implicó cambiar todo lo que estaba hecho, después de llevar unos meses de labor.
- Volver a programar en C++, pero de una forma más amplia. Este era un lenguaje con el que ya había tenido algún contacto, pero del que guardaba bastantes prejuicios, debido a una asignatura de la carrera, donde el programa en C++ que había que usar, causó grandes problemas a todos los que la cursaron.
- Aprender a trabajar con un grupo de investigadores, en una aplicación real. Esto implicaba que el proyecto aquí desarrollado, debía tener continuidad, y estar muy bien documentado. No era algo que se fuera a quedar sólo en los libros, como cualquier otra práctica que hubiera hecho anteriormente en la carrera. Por lo que este punto cobraba realmente mucha importancia en este caso, pasando a ser uno de los objetivos principales.
- Encapsular el nuevo software sobre una base existente, con unos procedimientos muy especializados, como es la arquitectura AD de Maggie. Esto requería de un estudio previo de esa arquitectura, para conocer el mecanismo de sus habilidades, eventos, y memoria a corto plazo. Y una vez hecho esto, desarrollar una habilidad especial, que hiciera de intermediaria de la arquitectura AD, con la conexión al tabletPC.
- Y además, los ordenadores debían de conectarse, para que trabajaran de forma distribuida y comunicarse mediante algún tipo de protocolo.

Pese a que todo esto representó verdaderamente un reto, también ha sido la mayor motivación que podía tener para realizar un proyecto de fin de carrera, con el que poner punto y final a mi carrera universitaria.

Desde siempre he sentido una gran vocación por la rama de inteligencia artificial, la cual tuve clara desde antes de empezar la universidad, y más al ver en el plan de estudios de informática, que se impartía una asignatura sobre la robótica. Hubo compañeros que me recomendaron no cursarla, por requerir de más esfuerzo que otras, pero eso no me amilanó.

El diseño gráfico, la animación y el modelado, por su creatividad y su componente visual, se convirtieron, más tarde, en el ámbito de la informática donde más satisfacción he encontrado con los resultados y el desarrollo de mi trabajo.

No obstante, aun todo este entusiasmo, tengo plena consciencia de que la robótica es un área con un impulso reducido en este país, con poca investigación fuera de los campus universitarios, y por lo tanto, de difícil acceso en un futuro, si deseo seguir este camino en el mundo laboral. Al igual que ocurre con el diseño gráfico.

Por todo ello, lo que en un inicio eran dificultades, pasaron a ser retos que superar con gusto, ya que suponían una gran oportunidad con escasa (aunque no nula), esperanza de que se pudiera repetir fuera de la universidad, y no debía de ser desaprovechada.

De esta forma:

- Se aprendió el uso de Flash 8, Flash CS4, de ActionScript 3.0 y sus versiones anteriores, además de ampliar conocimientos de Photoshop, como aplicación externa de edición de imágenes. Además de tener libertad creativa a la hora de diseñar la interfaz, pudiendo investigar el funcionamiento de distintos elementos de Flash, que en un principio, no se contemplaban dentro de este proyecto. Este es el caso de las secciones de la galería de imágenes, el reproductor de vídeo y de música, y del juego Sapiéntino que se decidió añadir como aportación personal al crecimiento de Maggie.
- Se ha profundizado en los conocimientos previos de C++, perdiendo la negatividad previa ante este lenguaje por completo, después del tiempo dedicado al mismo, en un marco positivo de trabajo.
- Se ha trabajado en el diseño de las secciones de la interfaz, y del programa servidor, de forma conjunta con el personal del laboratorio, para adaptarlo a sus investigaciones. También se ha tenido en cuenta su opinión respecto a la estructura de las clases, modificándola en ocasiones, para hacer más sencilla su comprensión por parte de los futuros usuarios.
- Se hizo un amplio estudio previo, de los requisitos de la arquitectura AD, y los patrones utilizados para eventos y memoria. Además, se contó con la ayuda incondicional prestada por las personas encargadas del mantenimiento del software de Maggie. Esto ayudó en gran medida para resolver las dudas que pudiera tener, de forma que la habilidad incluida en la arquitectura AD, se relaciona perfectamente con el resto de elementos y habilidades de Maggie, siguiendo las normas de diseño del laboratorio.

- Se ha afianzado el conocimiento anterior sobre la forma de trabajar con sockets y el desarrollo de aplicaciones distribuidas.

Finalmente, se puede concluir que a pesar de las dificultades, entre otras, debido al tiempo reducido que pude dedicar durante unos meses, la realización de este proyecto ha supuesto un gran aprendizaje, y una satisfacción personal, por haber tenido la ocasión de desarrollarlo.

Además, también ha servido para afianzar y demostrar los conocimientos adquiridos en la carrera. Un ingeniero en informática, debe de ser capaz de poder enfrentarse a cualquier lenguaje de programación, software, red o arquitectura, y mediante un período de adaptación, ser capaz de trabajar con ello, y con un equipo de trabajo, y encontrar la solución a los problemas que se presenten, sin olvidar nunca la importancia de los manuales de usuario, y de desarrollo. Ya que cualquier aplicación que se realice, es muy posible que ni siquiera sea utilizada por su autor, sino por terceras personas, que deben de estar perfectamente documentadas, para que el software proporcionado se comprenda y aproveche en su totalidad.

Un punto importante, que sin embargo, muchos informáticos solemos olvidar.

10 Referencias

- (1). **Aristóteles.** *República Ateniense*. 322 a.C.
- (2). **Zunt, Dominik.** *Karol Capek website*. [En línea] 2004.
<http://capek.misto.cz/english/index.html>. Consultada en junio de 2010.
- (3). **Byrd, Joseph S.** *Computers for mobile robots*. Columbia, University of South Carolina : s.n., 1993.
- (4). **Asimov, Isaac.** *Runaround*. Astounding Science Fiction. Marzo, 1942.
- (5). **Turing, Alan.** *On Computable Numbers, With an Application to the Entscheidungsproblem*. 1937.
- (6). **Turing, Alan.** *Computing machinery and intelligence*. Mind. 1950.
- (7). **Honda Motor Co.** *History of Robot Development Process*. [En línea] 2010.
<http://world.honda.com/ASIMO/history/>. Consultada en julio de 2010.
- (8). **MIT Artificial Intelligence Laboratory.** *Humanoid Robotics Group, Kismet Project*. [En línea] 2010.
<http://www.ai.mit.edu/projects/humanoid-robotics-group/kismet/kismet.html>.
 Consultada en julio de 2010.
- (9). **MIT NEWS.** *MIT News*. [En línea] 13 de Marzo de 1999.
<http://web.mit.edu/newsoffice/1999/cog.html>. Consultada en julio de 2010.
- (10). **Personal Robots Group, M.I.T. Media Lab.** [En línea] 2010.
<http://robotic.media.mit.edu/index.html>. Consultada en julio de 2010.
- (11). **Laboratorio de Robótica, Universidad Carlos III de Madrid.** *Docuwiki*. [En línea] 2010. <http://roboticslab.uc3m.es/dokuwiki/doku.php/>. Consultada en 2009/10.
- (12). **Adobe Systems Incorporated.** *Utilización de Adobe® Flash® CS4 Professional para Windows® y Mac OS*. California : s.n., 2008.
- (13). **Adobe Systems Incorporated.** *Programación con ActionScript™ 3.0*. California : s.n., 2007.

11 Bibliografía

Corrales, Ana. *Sistema de Identificación de Objetos Mediante RFID para un robot personal.* Leganés: s.n., 2007.

Carr, Dan. Flash Developer Center. *Controlling web video with ActionScript 3 FLVPlayback programming.* [En línea] 13 de Julio de 2009. http://www.adobe.com/devnet/flash/articles/flvplayback_programming.html. Consultada en mayo de 2010.

Castro Gonzalez, A., Tomas, E. y Salichs, M.A. *Gestión de estados básicos para el robot Maggie.* Leganés, Universidad Carlos III: s.n., 2008.

Castro Gonzalez, A. *Desde la teleoperación al control por tacto del robot Maggie.* Leganés, Universidad Carlos III de Madrid: s.n., 2008.

Adobe. *Adobe Flash Professional CS4.* [En línea] Junio de 2010. <http://www.adobe.com/es/products/flash/?promoid=BPBIQ>. Consultada a lo largo de 2009/10.

The C++ Resources Network. [En línea] 2010. <http://www.cplusplus.com/>. Consultada a lo largo de 2010.

Fedora. *Fedora.* [En línea] 2010. <http://fedoraproject.org/>. Consultada en marzo de 2010.

Salichs, M.A. *Apuntes de la asignatura de Robótica, de la licenciatura de Ingeniería en Informática.* Leganés, Universidad Carlos III de Madrid: s.n., 2009.

Museo Nacional Británico de Ciencia e Industria. *Web basada en la galería "Construyendo el Mundo Moderno" del museo.* [En línea] 2004. <http://www.makingthemodernworld.org.uk/>. Consultada en junio de 2010.

Computer History Museum. [En línea] 2010. <http://www.computerhistory.org/>. Consultada en julio de 2010.

RoboticsSpot. *Línea temporal sobre robótica.* [En línea] 2004. <http://www.roboticspot.com/especial/historia/his2004b.php>. Consultada en julio de 2010.

The History of Computing Project. *Timeline of Robotics.* [En línea] 17 de Marzo de 2010. <http://www.thocp.net/reference/robotics/robotics.html>. Consultada en julio de 2010.

Isom, James. *History of Robotics.* [En línea] 2005. <http://robotics.megagiant.com/history.html>. Consultada en julio de 2010.

Technology Guide. *TabletPC Review.* [En línea] 2010. <http://www.tabletpcreview.com/>. Consultada en agosto de 2010.

Vilchez, Ángel. *Tutorial: Tipos de Pantalla Táctil.* [En línea] 6 de Junio de 2009. <http://www.configurarequipo.com/doc1127.html>. Consultada en julio de 2010.

Nanoda. [En línea] 2010. www.nanoda.com. Consultada en septiembre de 2010

Comunidad Cristallab. *Comunidad de diseño web y desarrollo en internet.* [En línea] 2010. <http://www.cristallab.com/>. Consultada a lo largo de 2010.

Adobe Ibérica. *Canal Adobe.* [En línea]. <http://www.canaladobe.com/>. Consultada a lo largo de 2010.

Imagination Development. *Action Script 3 Articles and Tutorials.* [En línea] <http://actionscrip-blog.imaginationdev.com/>. Consultada a lo largo de 2010.

Mikeland.us. *Flash & ActionScript 3.0 Tutorials.* [En línea] <http://wordpress.mikeland.us/category/flash/>. Consultada a lo largo de 2010.

Republic of Code. *Flash ActionScript 3.0 Tutorials.* [En línea] <http://www.republicofcode.com/tutorials/flash/>. Consultada a lo largo de 2010.

Adobe Systems Incorporated. *Flash Help and Support.* [En línea] <http://www.adobe.com/support/flash/>. Consultada a lo largo de 2010.

Adobe. *Programación con Adobe ActionScript 3.0 para Adobe Flash.* [En línea] http://help.adobe.com/es_ES/ActionScript/3.0_ProgrammingAS3/. Consultada a lo largo de 2010.

The Flash-db.com Community. *The Flash-db tutorial, article and application directory.* [En línea] <http://www.flash-db.com/Tutorials/>. Consultada a lo largo de 2010.

RoboticsLab. *Web Principal del grupo de Robótica del Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III.* [En línea]. <http://roboticslab.uc3m.es/roboticslab/>. Consultada en junio de 2010.

Anexo: Implantación

En este capítulo se detalla el proceso a seguir para lanzar correctamente los programas cliente y servidor que forman este proyecto.

Pasos previos

Antes de comenzar, para que funcione la ejecución de Flash, se tienen que tener una versión de Flash Player 10, o superior, instalada en el PC. Se puede descargar de forma gratuita desde la siguiente URL:

<http://get.adobe.com/es/flashplayer/>

Para que la interfaz funcione en todos sus aspectos, deberá de ejecutarse el archivo publicado como *.exe, en un sistema operativo Windows XP, Vista, o 7.

Además, el fichero *.exe deberá de estar en la misma carpeta que el proyecto Flash, para respetar la estructura de ficheros. Esto es así, para mantener la misma ruta relativa con los XML, las imágenes, los iconos, videos y archivos de audio, que se cargan en el programa, ya que de otra forma, se producirán errores al tratar de leerlos. Es decir, la misma disposición del proyecto Flash, pero pudiendo prescindir de todos los archivos *.swf, *.fla y *.as que ya están compilados dentro de la publicación.

Esta aplicación se ha diseñado para el tabletPC actual del laboratorio, por lo que se ha colocado en el escrito una carpeta de nombre “PROYECTO FLASH”, con la copia en español y en inglés de la interfaz, junto a todos los archivos necesarios para su ejecución, en dos directorios por separado.

En caso de realizar modificaciones en el código de los ficheros, o en la interfaz Flash, se deberá de publicar todo el proyecto de nuevo para actualizarlas, desde Flash CS4 (ver 6.1.9 Creación de Proyectos y su Publicación). Si los cambios se limitan a los archivos XML, no será necesario.

Para ejecutar la interfaz, el tabletPC debe de tener red, estar encendido, y con el sistema operativo Windows iniciado, momento en el que se lanzará de forma automática el programa servidor de TightVNC ya instalado, para poder realizar la conexión vía escritorio remoto. Esto se hace debido a la ubicación de difícil acceso del PC.

Por otro lado, mediante un ordenador que esté conectado a la red del laboratorio, se debe de acceder al tabletPC con el cliente TightVNC Viewer. A través de la dirección IP y la contraseña, se puede acceder al escritorio, donde ya se podría ejecutar la interfaz Flash (ver Figura 60).

Con esto ya estaría listo el programa cliente diseñado en este proyecto.

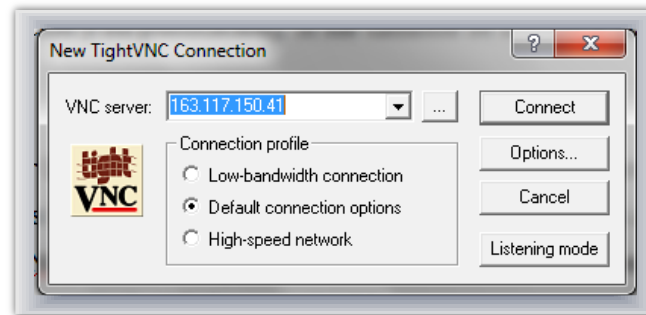


Figura 60. Cliente TightVNC Viewer.

Antes de lanzar el servidor, se debe de tener una cuenta en VisMaggie, y conectarse con un ordenador dentro de la red del laboratorio de robótica. Lo más sencillo es hacerlo por consola desde un sistema operativo Linux, con el comando SSH:

```
ssh -l usuario@163.117.150.33
```

En esa cuenta se debe de tener una versión del repositorio de la arquitectura, para poder probar y lanzar los archivos. Para descargar la rama del repositorio (este proceso necesitará tiempo), se deberá usar el siguiente comando:

```
svn co https://163.117.150.59/repoAD/branches/usuario/ carpetaDestino
```

Posteriormente, para que la compilación de un archivo del repositorio funcione correctamente, se debe añadir en el archivo oculto de Linux *bashrc*, la siguiente línea de código:

```
export AD_HOME=/home/rutaCarpetaDestinoRepositorio
```

De esta forma, se indicará donde se ha instalado la rama del repositorio, y se evitara problemas en la compilación con los makefiles.

Después, se deberá ir a la carpeta *tabletpc*, que es donde están los archivos que forman el servidor de este proyecto. Se encuentra dentro del directorio *skills*, en el que están todas las habilidades implementadas para Maggie.

Una vez aquí, para compilar, se podrá usar el archivo *makefile*, incluido para ese fin. Sin embargo, para que todo funcione correctamente, se deberá estar conectado en Vismaggie como root, para poder tener control sobre la arquitectura y los archivos de configuración y lanzar los servidores del hardware de Maggie. Si hay que iniciar la arquitectura, será con el script:

```
startAD.sh // accesible siendo root
```

Si en la información que aparecerá por consola, se detecta algún problema (sobre todo con los eventos, la memoria a corto plazo, o la habilidad de voz de Maggie, que es necesario para la ejecución), se deberá apagar, y activar de nuevo. Para pararla se deberá hacer con el script:

```
stopAD.sh // accesible siendo root
```

Nota: Para las cuentas a VisMaggie, el repositorio y las contraseñas, ponerse en contacto con el personal administrador del laboratorio.

Ejecución

Una vez realizados todos los pasos anteriores, se procederá a explicar el proceso para la correcta ejecución de la aplicación.

Primero, se debe de lanzar el servidor, para que esté disponible para recibir la petición de conexión del cliente. Para ello, en la carpeta *tabletpc*, del directorio *skills* de la rama del repositorio, con los archivos ya compilados, se deberá ejecutar por consola con el comando:

```
./runHabilidadFlash
```

Aparecerán unos mensajes que indicarán que se ha realizado la creación del socket, y que está a la espera de la entrada de un cliente.

Después, en la parte del tabletPC, bastará con abrir la carpeta que corresponda de la versión en español o en inglés, y ejecutando el archivo *.exe, se abrirá el programa (ver Figura 11). Con un único clic en cualquier punto de la pantalla (que no sea el botón de conexión, porque este paso habrá que darlo más tarde), se pondrá en pantalla completa, ajustándose a las medidas del hueco en Maggie.

Llegados a este punto, cliente y servidor ya se encuentran preparados. Para iniciar la comunicación, en la interfaz Flash se deberá pulsar el botón “Conectar”. Si hay algún error, aparecerá un mensaje en el tabletPC, dando aviso del mismo (ver Figura 12). Para volver a intentarlo, habrá que cerrar y volver a lanzar cliente y servidor.

Si no hay errores de conexión, aparecerá un menú en la consola donde se había ejecutado el servidor alojado en Vismaggie, donde habrá que elegir la opción 2 de “activar habilidad”. Esto hará que se cree el objeto correspondiente a la habilidad, y se lance un evento para la activación cíclica del método proceso. Una vez aquí, se habrá creado la comunicación tablet-Maggie satisfactoriamente, y se podrá utilizar toda la funcionalidad ofrecida en la aplicación.

Una vez finalizado el trabajo, se deberá pulsar el botón de cierre del menú principal de Flash, cerrar el servidor introduciendo un “3” por consola (opción de cierre del menú inicial), y bajar la arquitectura de Maggie, ejecutando el script:

```
stopAD.sh // accesible siendo root
```

Para terminar se debe poner a hibernar el tablet (hibernar, y no apagarlo, para que no se pierda la configuración de la pantalla), y salir de las cuentas de root y usuario en Vismaggie y terminar la conexión SSH.